COMMODORE 64

Guida di riferimento per il programmatore

COMMODORE 64

Guida di riferimento per il programmatore

Edizione 1984

INTRODUZIONE 1
* Che cosa contiene 2

* Come usare questa guida di riferimento 2

* Guida alle applicazioni del COMMODORE 64 4

* Rete informativa Commodore 7

1. REGOLE DELLA PROGRAMMAZIONE IN BASIC 1.1

* Introduzione 1.2

* Codici dello schermo video
(insieme dei caratteri BASIC) 1.2
Il Sistema Operativo(OS) 1.2

* Numeri e variabili della programmazione 1.4
Costanti intere, reali e stringa 1.4
Variabili intere, reali e stringa 1.6
Schiere intere, reali e stringa 1.7

* Espressioni e Operatori 1.8
Espressioni aritmetiche 1.8
Operazioni aritmetiche 1.9
Operatori relazionali 1.10
Operatori Logici 1.11
Gerarchia delle Operazioni 1.12
Operazioni su stringhe 1.13
Espressioni stringa 1.14

* Tecniche di programmazione 1.14
Conversione di dati 1.14
Uso dell'istruzione INPUT 1.15
Uso dell'istruzione GET 1.18
Compattazione dei programmi BASIC 1.20

2. VOCABOLARIO DEL LINGUAGGIO BASIC 2.1

* Introduzione 2.2

* Parole chiave, Abbreviazioni,
Tipi di Funzione del BASIC 2.3

* Descrizione delle parole chiave del BASIC 2.5

* Tastiera del COMMODORE 64 e sue caratteristiche 2.46

* Editor di Schermo 2.48

3. PROGRAMMAZIONE GRAFICA DEL COMMODORE 64 .... 3.1

* Panoramica sulla grafica 3.2
Modo Carattere del Video 3.2

Modo Bit Map 3.2
Animazioni 3.2

* Locazioni della Grafica 3.2
Selezione del Banco Video 3.3
Memoria di schermo 3.4
Memoria Colore 3.4
Memoria Carattere 3.4

* Modo Carattere Standard 3.7
Definizione del carattere 3.7

* Caratteri Programmabili 3.8

* Grafica del Modo Multicolore 3.13
Bit del Modo Multicolore 3.13

* Modo Colore di Fondo Esteso 3.16

* Grafica "Bit Map" 3.17
Modo Bit Map Standard ad Alta Risoluzione 3.18
Funzionamento 3.18

* Modo Bit Map Multicolore 3.21

* "Scrolling" rallentato 3.22

* Animazioni 3.24
Definizione di un'animazione 3.24
Puntatori dell'animazione 3.26
Attivazione delle animazioni 3.26
Disattivazione delle animazioni 3.27
Colori 3.27
Modo Multicolore 3.27
Impostazione di un'animazione nel Modo Multicolore 3.28
Animazioni ingrandite 3.28
Posizionamento delle animazioni 3.29
Riassunto del posizionamento delle animazioni 3.32
Priorita' di visualizzazione delle animazioni 3.33
Determinazione dei punti di contatto 3.33

* Altre caratteristiche della grafica 3.39
Azzeramento dello Schermo 3.39
Registro di Quadro 3.39
Registro di Stato dell'Interruzione 3.39
Combinazioni consigliate dei Colori di Schermo
e Carattere 3.40

* Programmazione delle Animazioni - Un ulteriore sguardo 3.41
Costruzione di Animazioni in BASIC
Un breve Programma 3.41
Compattazione dei Programmi di Animazioni 3.43
Posizionamento delle Animazioni sullo Schermo 3.44
Priorita' delle Animazioni 3.48
Come disegnare un'Animazione 3.48
Creazione di un'Animazione...minuto per minuto 3.49
Movimento dell'Animazione sullo Schermo 3.51

"Scrolling" verticale....3.51
Il Topolino Ballerino - Un esempio di Programma di Animazione....3.52
Tabella riassuntiva per la creazione di un'Animazione....3.59
Note per la creazione di un'Animazione....3.60

4. PROGRAMMAZIONE DI SUONI E MUSICA CON IL COMMODORE 64 ....4.1

* Introduzione....4.2
  Controllo del Volume....4.4
  Frequenze delle Onde Sonore....4.4

* Uso delle Voci Multiple....4.4
  Controllo delle Voci Multiple....4.7

* Modifica delle Forme D'Onda....4.8
  Introduzione alle Forme D'Onda....4.10

* Il Generatore d'Inviluppo....4.11

* Filtratura....4.13

* Tecniche Avanzate....4.15

* Sincronizzazione e Modulazione Circolare....4.19

5. DAL BASIC AL LINGUAGGIO MACCHINA....5.1

* Che cos'e' un linguaggio macchina?....5.2
  A cosa assomiglia il Codice Macchina?....5.3
  Semplice Mappa della Memoria del Commodore 64....5.3
  I Registri del Microprocessore 6510....5.4

* Come si scrivono i programmi in Linguaggio Macchina?....5.5
  64 MON....5.5

* Notazione Esadecimale....5.5
  La Prima Istruzione in Linguaggio Macchina....5.7
  Il Primo Programma....5.9

* Modi di Indirizzamento....5.10
  Pagina Zero....5.10
  Lo "Stack"....5.10

* Indicizzazione....5.11
  Indicizzato Indiretto....5.12
  Indiretto Indicizzato....5.12
  Salti e Controllo....5.13

* Sottoprocedure....5.14

* Suggerimenti utili per il Principiante....5.15

* Un Compito piu' impegnativo....5.16

* Insieme delle Istruzioni del Microprocessore

MCS 6510 - Sequenza Alfabetica 5.17
Modi di indirizzamento delle Istruzioni e relativi
Tempi di esecuzione 5.35

* Gestione della Memoria sul Commodore 64 .... 5.39

* Il KERNAL 5.47

* Attivita' di inizializzazione del KERNAL 5.48
Come usare il KERNAL 5.48
Routines del KERNAL richiamabili dall'Utente 5.50
Codici Errore 5.77

* Uso del Linguaggio Macchina da BASIC 5.78
Dove memorizzare le Routine in Linguaggio Macchina 5.79
Come si accede al Linguaggio Macchina 5.80

* Mappa della Memoria del Commodore 64 .... 5.82
Assegnazioni di INPUT/OUTPUT del Commodore 64 .... 5.88

6. GUIDA ALL'INPUT/OUTPUT 6.1

* Introduzione 6.2

* Output su TV 6.2

* Output su altri dispositivi 6.3
Output su stampante 6.4
Output su Modem 6.5
Uso dei registratori a cassetta 6.6
Memorizzazione dei Dati su Floppy Disk 6.7

* Porte Giochi 6.8
"Paddles" 6.10
Penna Ottica 6.11

* Descrizione dell'Interfaccia RS-232 .... 6.12
Schema Generale 6.12
Apertura di un Canale RS-232 .... 6.12
Prelievo di Dati da un Canale RS-232 .... 6.15
Invio di Dati ad un Canale RS-232 .... 6.16
Chiusura di un Canale Dati RS-232 .... 6.17
Esempi di Programmi in BASIC 6.20
Puntatori alla Locazione di Base del Buffer di
Ricezione/Trasmissione 6.21
Locazioni della Memoria di Pagina Zero ed uso
dell'Interfaccia di Sistema RS-232 .... 6.21
Locazioni della Memoria delle altre Pagine ed uso
dell'Interfaccia di Sistema RS-232 .... 6.21

* Porta Utente 6.22
Descrizione dei Pin di Porta 6.23

* Bus Seriale 6.25
Spinotti di Uscita del Bus Seriale 6.26

* Porta Espansione 6.29

* Cartuccia con Microprocessore Z-80 .... 6.31
Come usare il CP/M(R) Commodore 6.32
Elaborazione con il CP/M(R) Commodore 6.32

APPENDICE

A. Abbreviazioni delle parole chiave del BASIC 1
B. Codici dello Schermo Video 2
C. Codici ASCII e CHR$ 4
D. Mappe della Memoria Colore e dello Schermo 5
E. Valore delle Note Musicali 7
F. Bibliografia 9
G. Mappa dei Registri del Circuito VIC 11
H. Derivazione delle Funzioni Matematiche 13
I. Connessioni pin per Dispositivi di I/O 14
J. Conversione dei Programmi da BASIC standard al BASIC del COMMODORE 64 16
K. Messaggi di Errore 17
L. Specifiche del Circuito del Microprocessore 6510 19
M. Specifiche del Circuito CIA 6526 - Adattatore Interfaccia Complessa 31
N. Specifiche del Circuito 6566/6567 (VIC-II) 43
O. Specifiche del Dispositivo 6581 Interfaccia del Suono (SID) 58
P. Glossario 78

INDICE ANALITICO
PRONTUARIO DEL COMMODORE 64
SCHEMA ELETTRICO DEL COMMODORE 64

INTRODUZIONE

La GUIDA AL COMMODORE 64 e' stata sviluppata come strumento di lavoro e come guida di riferimento per coloro che vogliano sfruttare al massimo le notevoli possibilita' offerte dal COMMODORE 64. Questo Manuale contiene le informazioni necessarie alla programmazione, dall'esempio piu' semplice a quello piu' complesso. La GUIDA e' stata realizzata in modo tale che chiunque, dal programmatore dilettante in BASIC, al professionista esperto nel Linguaggio Macchina del 6502, possa trarre delle informazioni utili per sviluppare i propri programmi. Allo stesso tempo, questo libro mostra quanto realmente sia versatile il COMMODORE 64.

Lo scopo di questa GUIDA non e' l'insegnamento del linguaggio BASIC o del Linguaggio Macchina del 6502. Vi si trova comunque un ricco Glossario di termini ed una introduzione "quasi-guidata" a molte parti del libro. Se non si e' gia' in possesso di una discreta conoscenza del BASIC, e' consigliabile esaminare attentamente la Guida Utente del COMMODORE 64 di cui e' corredato il Computer. Tale Guida fornisce una facile introduzione al linguaggio BASIC. Se cio' nonostante si incontrassero ancora delle difficolta' nell'uso del BASIC, si rimanda all'esame della Bibliografia posta al termine di questo Volume (o all'Appendice N della Guida Utente).

La GUIDA AL COMMODORE 64 non e' altro che un riferimento; come per gran parte dei Manuali di Riferimento, la capacita' di applicare in modo creativo le sue informazioni dipende in effetti dal grado di conoscenza che si ha dell'argomento. In altre parole, un programmatore alle prime armi non e' in grado di usare tutti i dati e le informazioni contenute in questa GUIDA, finche' non abbia aumentato la propria esperienza di programmatore.

Questo libro puo' servire per ricavare un notevole bagaglio di preziose informazioni riguardanti la programmazione, scritte in un Italiano scorrevole e comprensibile, con la spiegazione dei termini tipici della programmazione. Il programmatore professionista vi potra' trovare, invece, tutte quelle informazioni necessarie per sfruttare al meglio le capacita' del COMMODORE 64.

CHE COSA CONTIENE

* Il "dizionario BASIC" include i comandi, le istruzioni e le funzioni del CBM 64 BASIC elencati in ordine alfabetico. E' stato realizzato un prontuario contenente tutti i termini e le relative abbreviazioni. Questo e' seguito da una sezione dedicata alla descrizione piu' dettagliata di ciascun termine, seguita a sua volta da esempi di programmi in BASIC che ne illustrano il modo di operare.
* Se si ha bisogno di un'Introduzione all'uso del Linguaggio Macchina con i programmi in BASIC, un aiuto iniziale viene fornito dalle illustrazioni dedicate ai principianti.
* Il KERNAL e' una potente caratteristica di tutti i computer prodotti dalla Commodore. Esso assicura che i programmi scritti oggi possano essere utilizzati anche sui Commodore di domani.
* La sezione riguardante la programmazione di I/O offre l'opportunita' di usare il computer al limite. Questa sezione descrive come collegare ed usare Penna Ottica, "Joystick", Unita' Disco, Stampanti e Dispositivi di Telecomunicazione chiamati Modem.
* Si puo' entrare nel mondo delle ANIMAZIONI, dei caratteri programmabili, della grafica ad alta risoluzione e delle piu' perfezionate e dettagliate immagini animate dell'industria dei microcomputer.
* Si puo' entrare anche nel mondo della sintesi musicale, creando canzoni ed effetti sonori con il miglior sintetizzatore disponibile su qualsiasi altro Personal Computer.
* Ai programmatori esperti, la sezione riguardante i linguaggi non residenti mette a disposizione le informazioni sulle capacita' del COMMODORE 64 di far "girare" linguaggi ad alto livello sotto CP/M (*). Questo in aggiunta al BASIC.

La Guida al COMMODORE 64 deve essere quindi considerata come un utile strumento d'aiuto, in grado di allietare le ore spese per la programmazione.

(*) CP/M e' un marchio registrato della Digital Research, Inc.)

COME USARE QUESTA GUIDA

In questo manuale vengono usate alcune notazioni convenzionali per descrivere la sintassi (la struttura delle frasi) dei comandi o delle istruzioni BASIC, e per illustrare, di ciascuna parola chiave del BASIC, sia la parte obbligatoria che quella facoltativa. Le regole per interpretare la sintassi delle istruzioni sono le seguenti:

1. Le parole chiave del BASIC sono visualizzate in lettere maiuscole. Esse devono comparire seguendo l'ordine dato nell'istruzione, inserendole esattamente come sono illustrate.
2. Le voci comprese fra i doppi apici ("") indicano dati variabili da introdurre. Sia i doppi apici che i dati al loro interno devono comparire come mostrato in ciascuna istruzione.

3. Le voci comprese tra parentesi quadrate ([]) indicano un parametro facoltativo delle istruzioni. Un parametro e' una limitazione o un aggettivo addizionale per le istruzioni. Ad ogni parametro facoltativo e' associato un dato. Inoltre, l'ellissi (...) indica che una voce facoltativa puo' essere ripetuta tante volte quante lo consente la linea di programma.
4. Se una voce fra parentesi quadrate e' SOTTOLINEATA, significa che DEVONO ESSERE USATI quei determinati caratteri nei parametri facoltativi inseriti esattamente come sono illustrati.
5. Le voci comprese fra parentesi acute (<>) indicano dati variabili che devono essere forniti dall'Utente. Mentre la barra (/)indica che si deve compiere una scelta fra due opzioni mutuamente esclusive.

ESEMPIO DI FORMATO DI SINTASSI:

```
OPEN<numero-file>,<dispositivo>[,<indirizzo>],
     "[<drive>:<nome-file>],[<modo>]"
```

ESEMPI DI ISTRUZIONI EFFETTIVE:

```
10 OPEN 2,8,6,"0:STOCK FOLIO,S,W"
20 OPEN 1,1,2,"CHECKBOOK"
30 OPEN 3,4
```

Quando si applicano le regole sintattiche ad una situazione pratica, la sequenza dei parametri nelle istruzioni puo' non essere quella mostrata negli esempi sintattici. Gli esempi non pretendono di mostrare tutte le possibili sequenze, ma soltanto i parametri necessari e quelli facoltativi.
Gli esempi di programmi di questo libro sono illustrati con le parole e gli operatori separati da spazi bianchi per una maggiore leggibilita'. Di solito il BASIC non richiede tale separazione, a meno che la loro omissione non possa creare ambiguita' o scorrettezze nella sintassi.
Di seguito sono illustrati alcuni esempi dei simboli usati per diversi parametri delle istruzioni nei capitoli seguenti. Tale lista non ha la pretesa di fornire ogni possibilita', ma di dare una miglior comprensione di come sono presentati gli esempi sintattici.

SIMBOLO	ESEMPIO	DESCRIZIONE
<numero-file>	50	Numero logico di un file
<dispositivo>	4	Numero di dispositivo hardware
<indirizzo>	15	Numero di indirizzo secondario di un dispositivo sul bus seriale
<drive>	0	Numero di disk drive fisico
<nome-file>	"TEST.DATA"	Nome di un file dati o programma
<costante>	"ABCDEFG"	Dato letterale fornito dal programmatore
<variabile>	X145	Qualunque nome o costante dei dati variabili del BASIC
<stringa>	AB$	Uso della variabile di tipo stringa richiesta
<numero>	12345	Uso della variabile di tipo numerico richiesta
<numero-linea>	1000	Numero di linea del programma attuale
<numerico>	1.5E4	Variabile intera o reale

GUIDA ALLE APPLICAZIONI DEL COMMODORE 64

Quando per la prima volta si e' pensato all'acquisto di un computer, ci si e' probabilmente chiesto: "Ora che mi posso permettere di comprare un computer, che cosa ci posso fare?". Il bello del COMMODORE 64 e' che pùo' realizzare tutto cio' che si vuole! Puo' essere utilizzato per calcolare e registrare sia il bilancio di casa che quello del lavoro, come elaboratore di testi, come compagno in tutta una serie di giochi d'azione, si puo' farlo cantare, si possono creare addirittura cartoni animati personali, ed altro ancora. Il bello dell'avere un COMMODORE 64 e' che, anche se facesse soltanto una delle cose elencate piu' avanti, i soldi sarebbero stati comunque spesi bene. Ma il 64 e' un computer completo, che fa proprio TUTTO quanto elencato, e quindi abbastanza!

Oltre a quanto detto si possono trovare numerose altre idee creative e pratiche iscrivendosi ad un Club locale per utenti Commodore ed abbonandosi alle riviste Commodore.

APPLICAZIONE	COMMENTI/REQUISITI
GIOCHI D'AZIONE	Si possono trovare i veri videogiochi Bally Midway come Omega Race, Gorf e Wizard of War, oppure i giochi "gioca ed impara" come Math Teacher I, Home Babysitter e Commodore Artist.
PUBBLICITA' COMMERCIALE	Collegando il COMMODORE 64 alla TV ed esponendolo in vetrina con un messaggio musicale animato, si puo' ottenere un forte richiamo per il negozio.
ANIMAZIONE	L'animazione grafica del COMMODORE 64 consente di creare veri cartoni animati ad otto differenti livelli, permettendo alle figure di muoversi liberamente.
BABYSITTER	La cartuccia HOME BABYSITTER del COMMODORE 64 tiene occupati per ore i bambini,insegnando nello stesso tempo l'associazione alfabeto/tastiera. Essa insegna anche particolari rapporti e concetti istruttivi.
PROGRAMMAZIONE IN BASIC	La GUIDA PER L'UTENTE DEL COMMODORE 64 e la serie di manuali e nastri IMPARA A PROGRAMMARE DA SOLO offrono un valido punto di partenza.
BUSINESS SPREADSHEET	Il COMMODORE 64 offre la Serie "Easy" di supporto al lavoro professionale che comprende il piu' potente WORD PROCESSOR ed il piu' vasto "spreadsheet" (foglio elettronico) disponibile per qualunque altro Personal Computer.
COMUNICAZIONI	Il COMMODORE 64 consente di entrare nell'affascinante mondo del collegamento in rete dei calcolatori. Se si collega un VICMODEM al

COMMODORE 64, si puo' comunicare con altri proprietari di computer in tutto il mondo.

COMPOSIZIONE DI CANZONI	Il COMMODORE 64 e' dotato del piu' sofisticato sintetizzatore musicale disponibile su qualsiasi altro computer. Possiede tre voci completamente programmabili, nove ottave e quattro forme d'onda controllabili. Sono a disposizione le Commodore Music Cartridges ed i Manuali Commodore Music, che aiutano a creare o riprodurre tutti i tipi di musica ed effetti sonori.
CP/M(*)	La Commodore mette a disposizione una cartuccia di facile montaggio per l'utilizzo del CP/M (*) e del software relativo. (*) CP/M e' un marchio registrato della Digital Research, Inc.
ESERCIZI DI ABILITA'	Il coordinamento mano/occhio, la destrezza manuale sono sollecitati dai numerosi giochi Commodore... tra cui "Jupiter Lander" e quelli di simulazione della guida notturna.
EDUCATIVO	Poiche', di per se' stesso, lavorare al computer e' educativo, il COMMODORE Educational Resource Book contiene le informazioni generali sugli usi didattici dei computer. Sono disponibili inoltre diverse cartucce didattiche realizzate allo scopo di insegnare un po' di tutto, dalla musica alla matematica, dalla astronomia all'arte.
LINGUA STRANIERA	Il set di caratteri programmabili del COMMODORE 64 consente di sostituire il set di caratteri standard con dei caratteri di una lingua straniera definiti dall'Utente.
GRAFICA E ARTE	Oltre alla Grafica Animata menzionata piu' sopra, il COMMODORE 64 permette di tracciare disegni ad alta risoluzione e a piu' colori, caratteri programmabili e le combinazioni di tutti i differenti modi di visualizzazione della grafica e dei caratteri.
STRUMENTO DI CONTROLLO	Il COMMODORE 64 ha una Porta Seriale, una Porta RS-232 ed una Porta Utente, per essere utilizzato in tutta una serie di speciali applicazioni industriali. Come optional, e' inoltre disponibile una cartuccia IEEE/488.
DIARI E SCRITTURA CREATIVA	Il COMMODORE 64 offrira' presto un eccezionale sistema di scrittura che uguaglia o supera la qualita' e la flessibilita' dei piu' costosi sistemi di scrittura disponibili. Ovviamente, si possono memorizzare le informazioni sia sul Disk Drive 1541 sia su registratore Datassette (TM), stampandole mediante una VICPRINTER o un PLOTTER.

CONTROLLO DELLA PENNA OTTICA	Le applicazioni che richiedono l'uso di una Penna Ottica si possono realizzare con una qualsiasi Penna Ottica che sia compatibile con il connettore della Porta Giochi (Game-Port) del COMMODORE 64.
PROGRAMMAZIONE IN LINGUAGGIO MACCHINA	La GUIDA AL COMMODORE 64 include una sezione sul Linguaggio Macchina, ed una sezione riguardante l'interfacciamento fra BASIC e codice macchina. Per uno studio piu' approfondito e' disponibile una bibliografia.
STAMPA DI MODULI E DI STIPENDI	Il COMMODORE 64 puo' essere programmato per trattare una quantita' di problemi commerciali basati sulle registrazioni a giornale. Lettere maiuscole/minuscole in unione con la grafica "Business Form" del COMMODORE 64 facilitano il disegno di moduli che successivamente possono essere stampati.
STAMPA	Il COMMODORE 64 puo' interfacciarsi con un grande numero sia di stampanti ad aghi e "letter quality", sia di plotter.
RICETTE	Tramite il COMMODORE 64 si possono registrare le ricette preferite su unita' di memoria a disco o a cassetta risolvendo cosi' il problema dei fogli disordinati, che spesso vengono persi quando se ne avrebbe piu' bisogno.
SIMULAZIONE	La simulazione per mezzo del computer consente di condurre esperimenti costosi o pericolosi ad un costo e rischio minimi.
SPORT	Source (TM) e Compuserve (TM) offrono entrambi informazioni sportive ottenibili sul COMMODORE 64 impiegando un VICMODEM.
QUOTAZIONI DI BORSA	Con un VICMODEM, ed iscrivendosi ad una rete di servizi adatta, il COMMODORE 64 diventa una telescrivente privata.

Queste sono solo alcune applicazioni per il COMMODORE 64. Come si puo' comprendere, per lavoro o per divertimento, a casa, a scuola o in ufficio, il COMMODORE 64 offre una soluzione pratica per qualsiasi tipo di necessita'. La Commodore tiene a far sapere che la sua assistenza agli Utenti INIZIA con l'acquisto di un computer Commodore. Ecco perche' si sono realizzate due pubblicazioni contenenti informazioni Commodore provenienti da tutto il mondo (in Canada e negli Stati Uniti esiste anche la possibilita' di collegamento da costa a costa ad una rete "bidirezionale" di "Computer Information")
Inoltre, incoraggiamo vivamente e sosteniamo in tutto il mondo la crescita dei Club di Utenti Commodore Essi sono un'ottima sorgente di informazioni per ogni proprietario di computer Commodore, dal principiante fino al piu' esperto. Le riviste descritte piu' avanti contengono le informazioni piu' aggiornate per entrare in contatto con il Club di Utenti piu' vicino.

Infine, il Rivenditore Commodore locale e' un'utile sorgente di assistenza ed informazione Commodore.

POWER/PLAY
The Home Computer Magazine

Quando si tratta di divertimento, di istruzione a domicilio, di applicazioni pratiche casalinghe, POWER/PLAY e' la prima sorgente di informazioni per gli utenti "casalinghi" Commodore. Scoprire dove si trovano i Club di Utenti piu' vicini e cosa stanno facendo, venire a conoscenza di software, giochi, tecniche di programmazione, telecomunicazioni, e novita'. POWER/PLAY e' il filo diretto personale con altri Utenti Commodore, centri di sviluppo esterni di software e hardware e la Commodore stessa. Pubblicazione trimestrale. Solo $10.00 per un anno di eccitante programmazione fra le mura di casa vostra.

COMMODORE
The Microcomputer Magazine

Ampiamente letto da insegnanti, uomini d'affari e studenti, cosi' come dai "computeristi", Commodore Magazine e' il veicolo principale per la diffusione di informazioni esclusive sull'uso piu' tecnico dei sistemi Commodore. Articoli pubblicati regolarmente su ogni numero si occupano di affari, scienza e didattica, suggerimenti per la programmazione, "riassunti da un taccuino tecnico", e molte altre caratteristiche di particolare interesse per chiunque usi o sia in procinto di acquistare un sistema Commodore per applicazioni commerciali, scientifiche o didattiche. COMMODORE e' il complemento ideale di POWER/PLAY .
Pubblicazione bimestrale. Abbonamento annuale: $15.00.

E PER ULTERIORI INFORMAZIONI...
...RIVOLGERSI ALLA NOSTRA RIVISTA SENZA CARTA

COMMODORE INFORMATION NETWORK

Ecco la rivista del futuro. Come supplemento ed ampliamento alle riviste POWER/ PLAY. e COMMODORE, la COMMODORE INFORMATION NETWORK - la nostra rivista "senza carta" - e' ora disponibile via telefono (negli Stati Uniti) per i computer Commodore usando un modem.
Entrando in uno dei nostri Computer Club, si puo' essere aiutati a risolvere un problema di computer, si puo' "parlare" con altri amici Commodore, oppure ottenere informazioni "fresche" sui nuovi prodotti e sulle risorse di software ed educative. In poco tempo si puo' essere addirittura in grado di risparmiarsi la fatica di digitare i programmi trovati su POWER/PLAY o COMMODORE, prelevandoli direttamente dalla Rete Informativa (un nuovo servizio per l'Utenza progettato per l'inizio del 1983). La cosa migliore e' che le risposte sono presenti anche prima che vengano formulate le domande (niente male, vero ?).
L'allacciamento alla nostra rivista elettronica richiede solamente un modem e l'abbonamento a Compuserve(TM), una delle piu' vaste reti di telecomunicazione degli Stati Uniti [in ogni pacchetto VICMODEM e' incluso un abbonamento annuale GRATUITO a Compuserve(TM)].

Per entrare in contatto con la Banca Dati Compuserve(TM) e' sufficiente comporre il numero telefonico e predisporre il modem. Al comparire sullo schermo del testo video Compuserve(TM), battere da tastiera G CBM. Quando compare l'indice del COMMODORE INFORMATION NETWORK (menu'), si puo' scegliere uno dei 16 argomenti, mettersi a proprio agio e godersi la rivista senza carta.
Ulteriori informazioni sono disponibili presso i Rivenditori Commodore, oppure contattando il Servizio Clienti Compuserve(TM) (800-848-8990; per l'Ohio 614-457-8600).

COMMODORE INFORMATION NETWORK

Descrizione Menu' Principale	Rivenditori Commodore
Codici Accesso Diretto	Risorse Educative
Comandi Speciali	Associazioni Utenti
Domande Utenti	Descrizioni
Public Bulletin Board	Domande e Risposte
Riviste e Aggiornamenti	Suggerimenti Software
Prodotti Annunciati	Suggerimenti Tecnici
Commodore News Direct	Directory Descriptions

CAPITOLO 1

regole della programmazione in BASIC

- Introduzione
- Codici dello Schermo Video (Insieme dei Caratteri BASIC)
- Numeri e variabili della Programmazione
- Espressioni ed Operatori
- Tecniche di Programmazione

INTRODUZIONE

Questo Capitolo tratta di come il BASIC memorizza e manipola i dati. Gli argomenti comprendono:

1) Un breve accenno alle componenti ed alle funzioni del Sistema Operativo e all'insieme di caratteri utilizzato dal COMMODORE 64.
2) La formazione di costanti e variabili. Quali sono i tipi di variabili. E come sono memorizzate le variabili e le costanti.
3) Le regole di calcolo aritmetico, test relazionali, trattamento di stringhe ed operazioni logiche. Sono comprese anche le regole per la composizione delle espressioni e le conversioni dei dati necessarie quando si usa il BASIC con diversi tipi di dati.

CODICI DELLO SCHERMO VIDEO
INSIEME DEI CARATTERI BASIC

IL SISTEMA OPERATIVO

Il Sistema Operativo risiede nei Circuiti di Memoria a Sola Lettura (ROM) ed e' una combinazione di tre moduli programma separati ma interdipendenti:

1) L'Interprete BASIC
2) Il KERNAL
3) L'Editor Video

1) All' Interprete Basic e' demandata la funzione di analizzare la sintassi delle istruzioni BASIC, e quella di realizzare i calcoli e/o il trattamento dei dati che gli vengono richiesti. L'Interprete BASIC ha un vocabolario di 65 "parole chiave" che assumono un particolare significato. I caratteri alfabetici maiuscoli e minuscoli e le cifre da 0 a 9 vengono usati per comporre sia parole chiave che nomi di variabili. Per l'Interprete hanno significato anche alcuni caratteri della punteggiatura e simboli speciali. La tabella 1.1 illustra i caratteri speciali ed il loro uso.
2) Il KERNAL tratta la maggior parte dei processi di controllo dei livelli di interruzione del sistema (per una trattazione dettagliata dei livelli di interruzione si rimanda al Capitolo 5). Il KERNAL controlla inoltre l'ingresso e l'uscita effettivi dei dati.
3) L'Editor Video controlla l'output sullo schermo video (televisore), e l'editazione del testo di un programma BASIC. Inoltre, l'Editor Video esamina l'input proveniente da tastiera cosi' da poter stabilire se i caratteri introdotti debbano essere eseguiti immediatamente, oppure inoltrati all'Interprete BASIC.

CARATTERE	NOME E DESCRIZIONE
	BLANK - Separa le parole chiave e i nomi delle variabili
;	PUNTO E VIRGOLA - Usato nelle liste di variabili per formattare l' output
=	UGUALE - Assegnazione di un valore ed impostazione di relazioni
+	PIU' - Addizione aritmetica o concatenazione di stringhe
-	MENO - Sottrazione aritmetica, operatore unario (-1)
*	ASTERISCO - Moltiplicazione aritmetica
/	BARRA - Divisione aritmetica
↑	FRECCIA IN ALTO - Elevamento aritmetico a potenza
(	PARENTESI APERTA - Valutazioni e funzioni di una espressione
)	PARENTESI CHIUSA - Valutazioni e funzioni di una espressione
%	PERCENTUALE - Dichiara un nome di variabile come intero
#	POUND - Precede il numero logico di un file in istruzioni di input/output
$	DOLLARO - Dichiara un nome di variabile come stringa
,	VIRGOLA - Usato nelle liste di variabili per formattare l'output; separa anche i parametri di comando
.	PUNTO - Punto decimale nelle costanti in virgola mobile
"	VIRGOLETTE - Racchiudono costanti stringa
:	DUE PUNTI - Separano piu' istruzioni BASIC su una linea
?	PUNTO INTERROGATIVO - Abbreviazione per la parola chiave PRINT
<	MINORE - Usato nei test relazionali
>	MAGGIORE - >> >> >>
π	PI GRECO - Costante numerica pari a 3.141592654

Tabella 1.1 - Insieme dei Caratteri CBM BASIC

Il Sistema Operativo fornisce due modi di operare in BASIC:

1) Modo DIRETTO
2) Modo PROGRAMMA

1) Usando il Modo DIRETTO le istruzioni BASIC non hanno il numero di linea all'inizio dell'istruzione. Esse vengono eseguite non appena viene premuto il tasto RETURN .
2) Il Modo PROGRAMMA e' quello usato per far girare i programmi. Usando questo modo, tutte le istruzioni BASIC devono avere all'inizio i numeri di linea. Si possono avere piu' istruzioni BASIC su una stessa linea, ma il loro numero e' limitato dal fatto che una linea logica di video non puo' avere piu' di 80 caratteri. Cio' vuol dire che se si supera il limite di 80 caratteri, occorre riportare l'istruzione BASIC che eccede tale lunghezza su una nuova linea, con un nuovo numero di linea.

Il COMMODORE 64 possiede due insiemi completi di caratteri che si possono usare sia da tastiera che da programma.
L'insieme 1 comprende le lettere alfabetiche maiuscole ed i numeri da 0 a 9, che possono essere battuti senza ricorrere al tasto SHIFT . Premendo il tasto SHIFT durante la digitazione, si utilizzano i caratteri grafici indicati sulla parte inferiore DESTRA dei

tasti. Tenendo premuto C= (tasto Commodore) durante la battitura, si utilizzano invece i caratteri grafici indicati sulla parte inferiore SINISTRA dei tasti. Tenendo premuto il tasto SHIFT ed un qualunque tasto sprovvisto di simboli grafici, si otterrà il simbolo indicato nella parte superiore del tasto.

L'insieme 2 comprende le lettere alfabetiche minuscole ed i numeri da 0 a 9, che possono essere battuti senza ricorrere al tasto SHIFT . Tenendo premuto questo tasto durante la battitura, si utilizzano le lettere alfabetiche maiuscole. Anche in questo caso, tenendo premuto il tasto C= (tasto Commodore), si utilizzano i simboli grafici indicati sulla parte SINISTRA inferiore dei tasti, mentre tenendo premuto il tasto SHIFT si utilizzano i simboli riportati nella parte superiore dei tasti sprovvisti di caratteri grafici.

Per passare da un insieme di caratteri all'altro, basta premere C= (tasto Commodore) e SHIFT contemporaneamente.

NUMERI E VARIABILI DELLA PROGRAMMAZIONE COSTANTI INTERE, REALI E STRINGA

Le COSTANTI sonq i valori che si inseriscono nelle istruzioni BASIC. Il BASIC usa questi valori per descrivere i dati durante l'esecuzione dell'istruzione. Il BASIC CBM puo' riconoscere ed elaborare tre tipi di dati:

1) NUMERI INTERI
2) NUMERI REALI
3) STRINGHE

Le COSTANTI INTERE sono numeri interi (senza punti decimali); devono essere comprese fra -32768 e +32767. LE COSTANTI INTERE NON DEVONO AVERE PUNTI DECIMALI O VIRGOLE FRA LE CIFRE. Se il segno piu' (+) e' tralasciato, la costante e' assunta come numero positivo. Gli zeri che precedono una costante vengono ignorati e non dovrebbero essere usati poiche' sprecano della memoria e rallentano il programma. Comunque, non provocano un errore. Gli interi sono memorizzati in forma binaria con 2 byte. Alcuni esempi di costanti intere sono:

```
   -12
  8765
-32768
   +44
     0
-32767
```

NOTA: NON inserire virgole all'interno del numero. Ad esempio, battere sempre 32000 anziche' 32,000. Se si inserisce una virgola all'interno di un numero si ottiene il messaggio di errore ?SYNTAX ERROR.

Le COSTANTI REALI sono numeri positivi o negativi, compresi i frazionari. Le parti decimali di un numero si possono visualizzare adoperando il punto decimale. Ancora una volta si ricordi che le virgole NON devono essere usate tra i numeri. Se il segno piu' (+) e' omesso, il COMMODORE 64 prende il numero come positivo. Se si tralascia il punto decimale, il computer considera il punto come se si trovasse dopo l'ultima cifra del numero. E come per gli interi, gli

zeri che precedono una costante vengono ignorati. Le costanti reali si possono utilizzare in due modi:

1) NOTAZIONE SEMPLICE
2) NOTAZIONE SCIENTIFICA

Le costanti reali vengono visualizzate sullo schermo fino alla nona cifra. Queste cifre possono descrivere valori compresi fra -999999999 e +999999999. Se si inseriscono piu' di nove cifre, il numero viene arrotondato alla decima cifra. Se questa cifra e' un numero maggiore o uguale a 5, avviene un arrotondamento per eccesso, altrimenti avviene un arrotondamento per difetto. Cio' puo' essere importante per i totali finali di alcuni numeri con cui ci si puo' trovare a lavorare. I numeri reali (memorizzati usando 5 bytes di memoria) vengono trattati nei calcoli usando una precisione di dieci cifre. Tuttavia, al momento della stampa dei risultati, i numeri sono arrotondati a nove cifre. Alcuni esempi di alcuni numeri reali semplici sono:

```
  1.23
 -.998877
+3.1459
  .7777777
-333.
  .01
```

I numeri piu' piccoli di .01 o piu' grandi di 999999999. sono stampati in NOTAZIONE SCIENTIFICA. Una costante reale in notazione scientifica e' composta da tre parti:

1) MANTISSA
2) LETTERA
3) ESPONENTE

La MANTISSA e' un numero reale semplice. La LETTERA E viene usata per indicare che il numero viene visualizzato in forma esponenziale. In altri termini, E rappresenta *10 (ad esempio, 3E3=3*10 3=3000). L'ESPONENTE indica per quale potenza di 10 il numero viene moltiplicato.
Sia la mantissa che l'eponente sono numeri con segno (+ o -). Il campo di definizione dell'esponente e' compreso fra -39 e +38 ed indica il numero di posizioni di cui il punto decimale nella mantissa verrebbe spostato a sinistra (-) o a destra (+) se il valore della costante fosse rappresentato come un numero semplice.
C'e' un limite per la grandezza dei numeri reali che il BASIC e' in grado di trattare, questo anche per la notazione scientifica: il numero piu' grande e' +1.70141183E+38, ed i calcoli che diano come risultato un numero maggiore di questo comportano il messaggio di errore **?OVERFLOW ERROR**. ; il numero reale piu' piccolo e' +2.93873588E-39 e, se i calcoli danno un risultato inferiore, questo viene assunto come zero e NON SI HA ALCUN MESSAGGIO DI ERRORE. Alcuni esempi di numeri reali in notazione scientifica ed i loro valori decimali sono:

```
235.988E-3    (.235988)
2359E6        (2359000000.)
-7.09E-12     (-.00000000000709)
-3.14159E+5   (-314159.)
```

Le COSTANTI STRINGA sono gruppi di informazioni alfanumeriche come lettere, numeri e simboli. Quando si digita una stringa da tastiera, la sua lunghezza non puo' superare lo spazio disponibile sulla linea di programma di 80 caratteri (cioe' tutto lo spazio NON occupato dal numero di linea e dal resto dell'istruzione).
Una costante stringa puo' contenere spaziature, lettere, numeri, punteggiature e caratteri di controllo del cursore o del colore in qualsiasi combinazione. Si possono addirittura inserire delle virgole fra i numeri. L'unico carattere che non puo' essere inserito in una stringa e' il doppio apice ("), in quanto questo carattere viene usato per definire l'inizio e la fine della stringa. Una stringa puo' assumere anche un valore nullo - cio' significa che non contiene alcun carattere. Si puo' trascurare il doppio apice al termine di una stringa se questa e' l'ultima parte di una linea o se e' seguita da due punti (:). Alcuni esempi di costanti stringa sono:

```
""   (Stringa nulla)
"L. 250.000"
"NUMERO DI IMPIEGATI"
```

NOTA: Per includere nelle stringhe il doppio apice si usa CHR$(34).

VARIABILI INTERE REALI E STRINGA

Le VARIABILI sono nomi che rappresentano i valori usati nelle istruzioni BASIC. Il valore rappresentato da una variabile puo' essere assegnato ponendo questa uguale ad una costante, oppure puo' essere il risultato dei calcoli del programma. I dati della variabile, come le costanti, possono essere numeri interi, reali o stringhe. Se si fa riferimento al nome di una variabile di un programma, prima che le sia stato assegnato un valore, l'interprete BASIC crea automaticamente la variabile assegnandole il valore zero, se questa e' un numero intero o reale, oppure il valore nullo, se e' una stringa.
I nomi delle variabili possono avere una lunghezza qualsiasi, ma solamente i primi due caratteri hanno significato per il CBM BASIC. Questo significa che tutti i nomi usati per le variabili NON devono avere i primi due caratteri uguali. I NOMI DELLE VARIABILI NON POSSONO ESSERE NE' CONTENERE PAROLE RISERVATE DEL BASIC. Le parole riservate includono tutti i comandi BASIC, le istruzioni, i nomi di funzione ed i nomi degli operatori logici. Se accidentalmente viene usata una parola riservata all'interno del nome di una variabile, sullo schermo viene visualizzato il messaggio ?SYNTAX ERROR
I caratteri usati per formare i nomi delle variabili sono quelli dell'alfabeto ed i numeri da 0 a 9. Il primo carattere del nome deve essere una lettera. I caratteri di dichiarazione del tipo di dato (% e $) possono essere usati come ultimo carattere del nome. Il segno di percentuale (%) indica che la variabile e' un intero, ed il simbolo di dollaro ($) dichiara una variabile stringa. Se non si usa un carattere di dichiarazione di tipo, l'interprete assume la variabile come reale. Alcuni esempi di nomi di variabili, assegnazioni di valori e tipi di dati sono:

```
A$="VENDITA    "     (Variabile stringa)
MTH$="GEN"+A$        (Variabile stringa)
```

```
K%=5            (Variabile intera)
CNT%=CNT%+1     (Variabile intera)
FP=12.5         (Variabile reale)
SOM=FP*CNT%     (Variabile reale)
```

SCHIERE INTERE REALI E STRINGA

Una SCHIERA e' una tabella (o lista) di dati associati a cui si puo' fare riferimento mediante un unico nome di variabile. In altre parole, una schiera e' una sequenza di variabili correlate. Per esempio, una tabella di numeri puo' essere vista come una schiera. I singoli numeri all'interno della tabella diventano gli "elementi" della schiera.

Le schiere sono un modo pratico e compatto di descrivere un grande numero di variabili correlate. Prendiamo ad esempio una tabellina di numeri, e supponiamo che abbia 10 righe di 20 numeri ciascuna, per un totale di 200 numeri. Se non si avesse la possibilita' di usare un unico nome per tutta la schiera, si dovrebbe assegnare un nome diverso a ciascun valore della tabellina. Invece, grazie alle schiere, occorre adoperare soltanto un nome per la schiera, mentre tutti i suoi elementi vengono identificati attraverso le posizioni che occupano al suo interno.

Le schiere possono essere di tipo intero, reale o stringa, e tutti gli elementi che le compongono sono del suo stesso tipo. Le schiere possono essere ad una dimensione (come una lista semplice) o a piu' dimensioni (come ad esempio una griglia contrassegnata con righe e colonne, o un Cubo di Rubik[R]). Si puo' identificare e fare riferimento univocamente ad ogni elemento di una schiera attraverso una variabile indice (o sottoscritto), racchiusa fra parentesi, che segue il nome della schiera.

Il massimo numero di dimensioni che, in teoria, puo' avere una schiera e' 255, ed il numero di elementi per ogni dimensione e' limitato a 32767. In pratica pero' le dimensioni delle schiere vengono limitate dallo spazio di memoria disponibile per registrare i loro dati e/o una linea logica di 80 caratteri dello schermo. Se una schiera ha soltanto una dimensione ed il suo indice non supera mai 10 (11 elementi: da 0 a 10), la schiera viene creata automaticamente dall'interprete e riempita di zeri (o di stringhe vuote se di tipo stringa) non appena si fa riferimento per la prima volta ad un qualsiasi elemento, altrimenti occorre usare l'istruzione BASIC DIM per definire struttura e dimensioni della schiéra. Si puo' determinare la quantita' di memoria necessaria per memorizzare una schiera in questo modo:

```
                 5 byte  per il nome della schiera
                +2 byte  per ogni dimensione della schiera
                +2 byte  per ogni elemento di tipo intero
        OPPURE  +5 byte  per ogni elemento di tipo reale
        OPPURE  +3 byte  per ogni elemento di tipo stringa
        E       +1 byte  per ogni carattere presente
                   in ciascun elemento di tipo stringa
```

Gli indici possono essere variabili e/o costanti intere, oppure un'espressione aritmetica che dia un risultato intero. Gli indici delle schiere a piu' dimensioni devono essere separati fra loro da virgole. Gli indici possono assumere dei valori che vanno da zero al

numero di elementi appartenenti ad ognuna delle dimensioni della schiera. Valori che superino tale intervallo provocano il messaggio d'errore **?BAD SUBSCRIPT.** Alcuni esempi di nomi di schiere, assegnazioni e tipi sono:

```
A$(0)="VENDITE"            (schiera stringa)
MTH$(K%)="GEN"             (schiera stringa)
G2%(X)=5                   (schiera intera)
CNT%(G2%(X))=CNT%(1)-2     (schiera intera)
FP(12*K%)=24.8             (schiera reale)
SOM(CNT%(1))=FP K%         (schiera reale)
```

A(5)=0	(imposta a 0 l'elemento 5 della schiera unidimensionale di nome "A")
B(5,6)=0	(pone uguale a 0 l'elemento situato nella posizione di riga 5 e colonna 6 della schiera bidimensionale di nome "B")
C(1,2,3)=0	(pone uguale a zero l'elemento situato nella posizione di riga 1, colonna 2 e profondita' 3 della schiera tridimensionale di nome "C")

ESPRESSIONI ED OPERATORI

Le espressioni sono formate usando costanti, variabili e/o schiere. Un'espressione puo' essere una singola costante, una variabile semplice o una variabile schiera di ogni tipo. Puo' anche essere una combinazione di costanti e variabili con operatori aritmetici, relazionali o logici, designate per produrre un singolo valore. Il funzionamento degli operatori verra' spiegato in seguito. Le espressioni possono essere distinte in due classi:

1) ARITMETICHE
2) STRINGA

Normalmente, le espressioni si pensano composte da due o piu' voci chiamate operandi. Al fine di produrre il risultato desiderato, ciascun operando e' separato da un singolo operatore. Cio' di solito viene fatto assegnando il valore di un'espressione ad un nome di variabile. Tutti gli esempi di costanti e variabili visti rapidamente erano anche esempi di espressioni.
Un operatore e' un simbolo speciale che l'interprete BASIC del COMMODORE 64 riconosce come rappresentante di un'operazione che deve essere eseguita sulle variabili o sui dati costanti. Uno o piu' operatori combinati con una o piu' variabili e/o costanti formano un'espressione. Il BASIC del COMMODORE 64 riconosce operatori logici, relazionali ed aritmetici.

ESPRESSIONI ARITMETICHE

Quando vengono risolte, le espressioni aritmetiche danno un valore intero o reale. Gli operatori aritmetici (+,-,*,/,)sono usati per eseguire addizioni, sottrazioni, moltiplicazioni, divisioni ed elevamenti a potenza, rispettivamente.

OPERAZIONI ARITMETICHE

Un operatore aritmetico definisce un'operazione aritmetica eseguita sui due operandi che si trovano ai lati dell'operatore. Le operazioni aritmetiche vengono eseguite usando numeri reali. Prima dell'esecuzione di un'operazione aritmetica, i numeri interi vengono convertiti in numeri reali. Il risultato viene di nuovo riconvertito in un intero se e' assegnato ad un nome di variabile intera.

ADDIZIONE (+): Il segno piu' (+) indica che l'operando sulla destra e' addizionato all'operando sulla sinistra.

ESEMPI:

```
2+2
A+B+C
X%+1
BR+10E-2
```

SOTTRAZIONE (-): Il segno meno (-) indica che l'operando di destra viene sottratto da quello di sinistra.

ESEMPI:

```
4-1
100-64
A-B
55-142
```

Il segno meno puo' essere usato anche come operatore unario. Cio' significa che, posto davanti ad numero, fa interpretare quest'ultimo come numero negativo. In altre parole, e' come se il numero venisse sottratto da zero.

ESEMPI:

```
-5
-9E4
-B
4-(-2)    equivalente a 4+2
```

MOLTIPLICAZIONE(*): Un asterisco indica che l'operando di sinistra viene moltiplicato per l'operando di destra.

ESEMPI:

```
100*2
50*0
A*X1
R%*14
```

DIVISIONE (/): La sbarra specifica che l'operando di sinistra viene diviso per quello di destra.

ESEMPI:

```
10/2
6400/4
A/B
4E2/XR
```

ELEVAMENTO A POTENZA (↑): La freccia verso l'alto indica che

l'operando sulla sinistra e' elevato alla potenza specificata dall'operando di destra (esponente). Se l'operando sulla destra e' 2, allora l'operando sulla sinistra e' elevato al quadrato, se e' 3 al cubo, ecc. Affinche' il risultato dell'operazione dia un numero reale valido, l'esponente puo' essere un numero qualsiasi.

ESEMPI:

2↑2	equivalente a 2*2
3↑3	equivalente a 3*3*3
4↑4	equivalente a 4*4*4*4
AB↑CD	
3↑−2	equivalente a ⅓*⅓

OPERATORI RELAZIONALI

Gli operatori relazionali (<,=,>,<=,=>,<>) sono usati principalmente per confrontare i valori di due operandi, ma producono anche un risultato aritmetico. Quando si usano in operazioni di confronto, gli operatori relazionali e gli operatori logici (AND, OR, NOT) producono la valutazione aritmetica vero/falso di un'espressione. In un'espressione, se il rapporto e' vero al risultato viene assegnato il valore -1, mentre se e' falso il risultato e' 0. Gli operatori relazionali sono i seguenti:

<	MINORE
=	UGUALE
>	MAGGIORE
<=	MINORE O UGUALE
>=	MAGGIORE O UGUALE
<>	DIVERSO

ESEMPI:

1=5−4	vero (-1)
14>66	falso (0)
15>=15	vero (-1)

Gli operatori relazionali possono essere utilizzati per il confronto di stringhe. In questo caso, l'ordinamento delle lettere dell'alfabeto e' A<B<C<D ecc. Le stringhe sono confrontate valutando la relazione tra caratteri corrispondenti, muovendo da sinistra a destra (vd. Operazioni su Stringhe).

ESEMPI:

"A" < "B"	vero (-1)
"X" = "YY"	falso (0)
BB$ <> CC$	

I dati numerici possono essere confrontati solamente con altri dati numerici, cosi' come le stringhe posono essere confrontate solamente con altre stringhe; in ogni altro caso viene generato il messaggio di errore **?TYPE MISMATCH** . Gli operandi numerici che devono essere confrontati vengono innanzitutto convertiti dalla forma intera a quella reale. Dopodiche' si confrontano i valori reali per attribuire loro il risultato di vero o falso.
Al termine di ogni confronto si ottiene un numero intero,

indipendentemente dal tipo di dato dell'operando (anche nel caso che siano entrambi stringhe). Di conseguenza, il confronto fra due operandi puo' essere usato come operando durante l'esecuzione dei calcoli. Il risultato e' 0 oppure -1, e puo' essere usato in qualunque modo eccetto che come divisore, dato che la divisione per zero non ha significato.

OPERATORI LOGICI

Gli OPERATORI LOGICI (AND, OR, NOT) possono essere usati per modificare i significati degli operatori relazionali, o per produrre un risultato aritmetico. Gli operatori logici possono dare risultati diversi da 0 e -1, anche se qualsiasi risultato diverso da zero viene assunto come vero, quando si testi una condizione vera/falsa.

Gli operatori logici (chiamati talvolta Operatori Booleani) possono anche essere usati per eseguire operazioni logiche su due operandi, di cui viene considerata una sola cifra binaria (bit). Quando si usa l'operatore NOT, l'operazione viene eseguita sul solo operando di destra. Gli operandi devono essere compresi fra -32768 e +32767 (i numeri reali vengono convertiti in interi), e le operazioni logiche devono dare un risultato intero.

Le operazioni logiche sono eseguite bit per bit su due operandi. La AND logica da' come risultato 1 solo se entrambi i bit rispettivi degli operandi sono a 1. La OR logica da' come risultato 1 se uno dei bit degli operandi e' uguale a 1. La NOT logica da' come risultato il valore opposto di ciascun bit di un singolo operando. In altre parole, e' come dire "Se e' NOT 1 allora e' 0, se e' NOT 0 allora e' 1".

La OR esclusiva (XOR) non ha operatore logico, ma viene eseguita come parte dell'istruzione WAIT. La OR esclusiva significa che se i bit dei due operandi sono uguali allora il risultato e' 0, altrimenti e' 1.

Le operazioni logiche sono definite da gruppi di istruzioni, che insieme costituiscono la "Tavola della Verita'" booleana, come mostrato nella tabella 1.2.

Il risultato dell'operazione AND e' 1 solo se entrambi i bit sono uguali a 1:

1 AND 1 = 1
0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

Il risultato della OR e' 1 solo se almeno un bit e' 1:

1 OR 1 = 1
0 OR 1 = 1
1 OR 0 = 1
0 OR 0 = 0

NOTA: segue il complemento logico di ciascun bit:

NOT 1 = 0
NOT 0 = 1

La OR esclusiva (XOR) fa parte dell'istruzione WAIT:

1 XOR 1 = 0
1 XOR 0 = 1
0 XOR 1 = 1
0 XOR 0 = 0

Tabella 1.2 - Tavola booleana della Verità

Gli operatori logici AND, OR, NOT specificano che, nelle espressioni a due operandi, un'operazione di aritmetica booleana deve essere eseguita da tutte e due le parti dell'operatore. Solamente nel caso di NOT viene preso in considerazione il solo operando di destra. Le operazioni logiche (o di aritmetica booleana) non vengono eseguite finche' non siano state completate tutte le operazioni aritmetiche e logiche di un'espressione.

ESEMPI:

```
IF A=100 AND B=100 THEN 10 (Se sia A che B hanno valore 100 allora il
                            risultato e' vero)

A=96 AND 32:PRINT A (A=32)

IF A=100 OR B=100 THEN 20 (Se A o B sono uguali a 100 allora il
                           risultato e' vero)

A=64 OR 32:PRINT A        (A=96)

IF NOT X<Y THEN 30        (Se X=>Y allora il risultato e' vero)

X=NOT 96                  (Il risultato e' -97 [complemento a 2])
```

GERARCHIA DELLE OPERAZIONI

Tutte le espressioni eseguono tipi diversi di operazioni, in base alla gerarchia fissata. In altri termini, alcune operazioni vengono eseguite prima di altre. Il normale ordine delle operazioni puo'

essere modificato racchiudendo due o piu' operandi fra parentesi (), creando cosi' una "sottoespressione". In questo modo, prima viene calcolato il valore dentro le parentesi, poi quello al di fuori.

Usando le parentesi, si deve fare attenzione che il numero delle parentesi di destra sia uguale a quello delle parentesi di sinistra. Se alcune parentesi rimangono aperte, compare il messaggio BASIC **?SYNTAX ERROR**

Inserendo dentro le parentesi gruppi di operandi racchiusi essi stessi fra parentesi, si possono formare delle espressioni a piu' livelli. Questo procedimento prende il nome di nidificazione. Le parentesi possono essere nidificate fino ad un massimo di 10 livelli. L'espressione che si trova al livello piu' basso viene eseguita per prima. Alcuni esempi di espressioni sono:

```
A+B
C↑(D+E)/2
((X-C↑(D+E)/2)*10)+1
GG$>HH$
JJ$+"MORE"
K%=1 AND M<>X
K%=2 OR (A=B AND M<X)
NOT (D=E)
```

Normalmente, l'interprete BASIC realizza le operazioni su espressioni eseguendo prima le operazioni aritmetiche, poi le operazioni relazionali ed infine le operazioni logiche. Sia gli operatori logici che quelli aritmetici hanno un proprio ordine di precedenza (o gerarchia di operazione). D'altra parte, gli operatori relazionali non hanno un ordine di precedenza, e vengono eseguiti da sinistra a destra durante la valutazione dell'espressione.

Se tutti gli operatori rimanenti hanno lo stesso livello di precedenza, le operazioni vengono eseguite da sinistra a destra. Nelle espressioni con le parentesi, viene ugualmente mantenuto l'ordine di precedenza. La gerarchia delle operazioni logiche ed aritmetiche dalla prima all'ultima, in ordine di precedenza, e' mostrata nella Tabella 1.3.

OPERATORE	DESCRIZIONE	ESEMPIO
↑	Elevamento a potenza	BASE.EXP
-	Negazione (Meno Unario)	-A
*/	Moltiplicazione/Divisione	AB*CD, EF/GH
+-	Addizione/Sottrazione	CNT+2, JK-PQ
>=<	Operazioni Relazionali	A <= B
NOT	NOT logico (complemento intero a 2)	NOT K%
AND	AND logico	JK AND 128
OR	OR logico	PQ OR 15

Tabella 1.3 - Gerarchia delle operazioni su espressioni

OPERAZIONI SU STRINGHE

Le stringhe vengono confrontate utilizzando gli stessi operatori relazionali (=, <>, <=, =>, <, >) utilizzati per confrontare i numeri. I confronti di stringhe vengono fatti prendendo un carattere alla volta (da sinistra a destra) da ciascuna stringa, e valutando ciascuna posizione del codice del carattere prelevato dal set di caratteri

PET/CBM. Se i codici carattere sono uguali, allora anche i caratteri sono considerati uguali. Se i codici carattere sono diversi, il carattere con il numero di codice piu' basso e' il piu' basso nell'insieme dei caratteri. I confronti terminano quando viene raggiunta la fine dell'una o dell'altra stringa. Se tutti i codici sono uguali, viene considerata minore la stringa piu' corta. GLI SPAZI BIANCHI CHE UNA STRINGA SI PORTA DIETRO SONO SIGNIFICATIVI.

Alla fine di tutti i confronti, indipendentemente dal tipo dei dati, si ottiene un risultato intero. Cio' e' vero anche se entrambi gli operandi sono stringhe. Inoltre, il confronto di due operandi stringa puo' essere usato come operando nell'esecuzione di calcoli. Il risultato e' -1 o 0 (vero o falso, rispettivamente), e puo' essere usato in qualunque modo eccetto che come divisore, poiche' la divisione per zero non ha significato.

ESPRESSIONI STRINGA

Le espressioni sono trattate come se fossero seguite da un implicito "<>0". Cio' significa che se un'espressione risulta vera, allora vengono eseguite le istruzioni BASIC che la seguono, sulla stessa linea di programma. Se invece l'espressione e' falsa, il resto della linea e' ignorato, e viene eseguita la linea di programma successiva.

Come per i numeri, le operazioni possono essere eseguite anche su variabili stringa. L'unico operatore aritmetico stringa riconosciuto dal BASIC CBM e' il segno piu' (+), usato per eseguire le concatenazioni delle stringhe. Quando si concatenano le stringhe, la stringa alla destra del segno piu' viene aggiunta alla stringa di sinistra, ottenendo una terza stringa, che puo' essere stampata subito, usata in un confronto oppure assegnata ad nome di variabile. Se si confronta una stringa con un numero, o viceversa, compare il messaggio BASIC ?TYPE MISMATCH. I seguenti sono esempi di concatenazioni ed espressioni stringa:

```
10 A$="NOME":B$="FILE"
20 NAM$=A$+B$              (da' la stringa NOMEFILE)
30 RES$="NUOVO "+A$+B$ (da' la stringa NUOVO NOMEFILE)
             ^
             |_____ Si noti lo spazio
```

TECNICHE DI PROGRAMMAZIONE

CONVERSIONE DEI DATI

Quando e' necessario, l'interprete CBM BASIC converte un dato numerico da intero a reale o viceversa, rispettando le seguenti regole:

* Tutte le operazioni aritmetiche e relazionali sono eseguite in virgola mobile. Per valutare l'espressione, gli interi vengono convertiti in reali, ed il risultato viene riconvertito in intero.
* Se il nome di una variabile numerica di un certo tipo e' posto uguale ad un dato numerico di tipo diverso, il numero viene convertito e memorizzato nello stesso modo del tipo di dato

dichiarato nel nome della variabile.

* Quando un valore reale viene convertito ad intero, la parte frazionaria viene troncata, ed il risultato intero risulta minore o uguale al valore in virgola mobile. Se il risultato esce dall'intervallo che va da -32768 a +32767, compare il messaggio BASIC: ?ILLEGAL QUANTITY

USO DELL'ISTRUZIONE INPUT

Una volta conosciute le variabili, il passo successivo consiste nel loro abbinamento all'istruzione INPUT per applicazioni pratiche della programmazione.

Come primo esempio, si puo' pensare ad una variabile come ad una "cella di memoria" dove il COMMODORE 64 memorizza la risposta ad una domanda dell'Utente. Se si vuole che un programma chieda all'Utente di digitare un nome, tale nome puo' essere assegnato alla variabile N$. In questo modo, ogni volta che nel programma si scrive PRINT N$, il COMMODORE 64 stampa automaticamente il nome digitato dall'Utente.

Per prima cosa, digitare sul COMMODORE 64 la parola NEW, premere il tasto RETURN e battere questo esempio:

```
10 PRINT"COME TI CHIAMI ?":INPUT N$
20 PRINT"CIAO, "N$"!"
```

In questo esempio si usa N per ricordare che in questa variabile e' contenuto il nome. Il segno del dollaro ($) e' usato per indicare al computer che si sta usando una variabile stringa. Risulta molto importante la differenza fra questi due tipi di variabile:

1) NUMERICA
2) STRINGA

Si ricordera' dai primi paragrafi che le variabili numeriche sono usate per memorizzare valori numerici come 1, 100, 4000, ecc. Una variabile numerica puo' essere rappresentata da una lettera (A), da due lettere (AB), da una lettera ed un numero (A1), oppure da due lettere ed un numero (AB1). L'uso di nomi corti consente di risparmiare memoria. Un altro aiuto si ha utilizzando nello stesso programma lettere e numeri di categorie diverse (A1, A2, A3). Inoltre, se da una risposta si desiderano ricevere numeri interi anziche' decimali, basta ggiungere il segno di percentuale alla fine del nome della variabile (AB%, A1%, ecc.).

Consideriamo ora alcuni esempi riportanti diversi tipi di variabili ed espressioni unite all'istruzione INPUT:

```
10 PRINT "BATTI UN NUMERO":INPUT A
20 PRINT A

10 PRINT "BATTI UNA PAROLA":INPUT A$
20 PRINT A$

10 PRINT "BATTI UN NUMERO":INPUT A
20 PRINT A"MOLTIPLICATO 5 UGUALE"A*5
```

NOTA: Il terzo esempio mostra che i MESSAGGI vengono inseriti fra i doppi apici (""), mentre le variabili sono al di fuori. Si puo' inoltre notare, nella riga 20, che la variabile A viene stampata

prima del messaggio "MOLTIPLICATO 5 UGUALE", e che per ultimo viene eseguito il calcolo A*5 (moltiplica per 5 la variabile A)

Nella maggior parte dei programmi, i calcoli sono molto importanti. Si puo' scegliere se usare "numeri effettivi" o variabili, ma se si sta lavorando con numeri forniti dall'Utente ci si deve ricordare di sostituirli con variabili numeriche. Iniziamo chiedendo all'Utente di battere due numeri; un modo e' il seguente:

```
10 PRINT"BATTI DUE NUMERI":INPUT A:INPUT B
```

ESEMPIO DI BILANCIO DI ENTRATA/USCITA

```
5 PRINT "♥"  ← SHIFT CLR/HOME
10 PRINT"MONTHLY INCOME":INPUT IN
20 PRINT
30 PRINT"EXPENSE CATEGORY 1":INPUT E1$
40 PRINT"EXPENSE AMOUNT":INPUT E1
50 PRINT
60 PRINT"EXPENSE CATEGORY 2":INPUT E2$
70 PRINT"EXPENSE AMOUNT":INPUT E2
80 PRINT
90 PRINT"EXPENSE CATEGORY 3":INPUT E3$
100 PRINT"EXPENSE AMOUNT":INPUT E3
110 PRINT "♥"  ← SHIFT CLR/HOME
120 E=E1+E2+E3
130 EP=E/IN
140 PRINT"MONTHLY INCOME: $"IN
150 PRINT"TOTAL EXPENSES: $"E
160 PRINT"BALANCE EQUALS  $"IN-E
170 PRINT
180 PRINT E1$"="(E1/E)*100"% OF TOTAL EXPENSES"
190 PRINT E2$"="(E2/E)*100"% OF TOTAL EXPENSES"
200 PRINTE3$"="(E3/E)*100"% OF TOTAL EXPENSES"
210 PRINT
220 PRINT"YOUR EXPENSES="EP*100"% OF YOUR TOTAL
INCOME"
230 FOR X=1TO5000:NEXT:PRINT
240 PRINT"REPEAT? (Y OR N)":INPUT Y$:IF Y$="Y"THEN5
250 PRINT "♥":END
          ↖ SHIFT CLR/HOME
```

NOTA: IN NON puo' essere uguale a zero; E1, E2, E3 NON possono essere tutti uguali a zero contemporaneamente.

SPIEGAZIONE LINEA PER LINEA DEL BILANCIO ENTRATA/USCITA

Linea	Descrizione
5	Esegue la pulizia dello schermo
10	Istruzione PRINT/INPUT
20	Inserisce una linea
30	Categoria di Spesa 1 = E1$
40	Ammontare della spesa 1 = E1
50	Inserisce una linea
60	Categoria di Spesa 2 = E2$
70	Ammontare della spesa 2 = E2
80	Inserisce una linea
90	Categoria di Spesa 3 = E3$
100	Ammontare della spesa 3 = E3
110	Esegue la pulizia dello schermo
120	Somma l'ammontare delle spese = E
130	Calcola la percentuale Uscita/Entrata
140	Visualizza l'Entrata
150	Visualizza il Totale Uscite
160	Visualizza le Entrate e le Uscite
170	Inserisce una linea
180-200	Calcolano qual e' la percentuale di ogni Uscita rispetto al Totale Uscite
210	Inserisce una linea
220	Visualizza la percentuale Entrata/Uscita
230	Ciclo di ritardo

Moltiplichiamo ora quei due numeri per generare una nuova variabile C, come e' riportato nella seguente linea 20:

```
20 C=A*B
```

e stampiamo il risultato sotto forma di un messaggio:

```
30 PRINT A"MOLTIPLICATO"B"UGUALE"C
```

Digitiamo queste tre linee e "facciamo girare" (RUN) il programma. Notare che i messaggi sono racchiusi fra i doppi apici, mentre le variabili no.

Poniamo ora il segno del dollaro ($) davanti al numero rappresentato dalla variabile C. Tale segno deve essere stampato dentro gli apici e davanti alla variabile C. Per aggiungere il $ nel programma premere i tasti RUN/STOP e RESTORE, quindi digitare la linea 40 come segue:

```
40 PRINT "$" C
```

Premere ora RETURN, e poi RUN seguito di nuovo da RETURN. Il segno del dollaro deve essere posto tra apici, in quanto la variabile C rappresenta solo un numero, e non puo' contenere un $. Se C contiene il valore 100, il COMMODORE 64 visualizza $100. Ma se si prova a scrivere PRINT $C viene emesso il messaggio ?SYNTAX ERROR.

Un ultimo accenno a $: si puo' anche creare una variabile che rappresenta il segno del dollaro, e che puo' essere sostituita al simbolo $ che si vuole usare. Per esempio:

```
10 Z$="$"
```

A questo punto, quando si ha bisogno di inserire un $, si puo' usare la variabile stringa Z$:

```
10 Z$="$":INPUT A
20 PRINT Z$A
```

La riga 10 definisce il simbolo $ come una variabile stringa di nome Z$; inoltre, richiede un numero chiamato A. La linea 20 stampa Z$ ($) accanto ad A (numero).

Probabilmente, risulta piu' facile assegnare a variabili stringa certi caratteri, come $, piuttosto che digitare "$" ogni volta che si voglia calcolare una cifra in dollari o qualunque altra segno richiedente i doppi apici, come %.

USO DELL'ISTRUZIONE GET

La maggior parte dei programmi semplici usa l'istruzione INPUT per accettare i dati inviati dall'Utente che opera sul computer. Quando si manifestano esigenze più complesse, come la protezione dagli errori di battitura, l'istruzione GET permette di operare in modo più flessibile, rendendo i programmi più "intelligenti". In questo paragrafo si illustra l'uso dell'istruzione GET per permettere ai vari programmi di sfruttare alcune caratteristiche particolari dell'editing di schermo.

Il COMMODORE 64 ha un buffer di tastiera che può contenere fino a 10 caratteri. Ciò significa che se il computer è occupato in alcune operazioni quindi non è in grado di ricevere messaggi dalla tastiera (in altri termini, la tastiera è disabilitata), si può lo stesso

continuare ad introdurre fino a 10 caratteri, che verranno usati non appena il computer ha terminato il suo lavoro. Quanto esposto è dimostrabile provando a digitare:

```
10 TI$="000000"
20 IF TI$ < "000015" THEN 20
```

Digitare ora RUN e premere RETURN ; quindi, mentre il programma sta "girando", proviamo a digitare la parola HELLO.
Si puo' notare che non accade nulla per circa 15 secondi dall'inizio del programma. Trascorso questo tempo, sullo schermo compare il messaggio HELLO.
Supponiamo di essere in coda davanti ad un cinema. La prima persona in coda e' la prima a prendere il biglietto ed a lasciare la coda, mentre l'ultima persona in coda e' l'ultima a prendere il biglietto. L'istruzione GET si comporta come il bigliettaio: per prima cosa si accerta che ci siano dei caratteri "in coda", cioe' che sia stato battuto qualche tasto. Se la risposta e' affermativa, il carattere incontrato prende posto nella variabile appropriata; se la risposta e' negativa, alla variabile viene assegnato il valore vuoto.
A questo punto, e' importante notare che se si prova ad inserire nel buffer piu' di 10 caratteri alla volta, tutti quelli dopo il decimo sono persi.
Poiche' l'istruzione GET e' attiva anche quando non viene inserito alcun carattere, risulta necessario inserire tale istruzione in un ciclo, in modo da farle attendere la prima battitura di un tasto o il primo carattere inviato da programma.
La forma consigliata per l'istruzione GET e' la seguente (digitare NEW per cancellare il programma precedente):

```
10 GET A$ : IF A$ = "" THEN 10
```

Si noti che non ci sono spazi fra gli apici: cio' indica un valore vuoto, che rimanda il programma indietro all'istruzione GET, con un ciclo ("loop") continuo, finche' non viene premuto un tasto del computer. Una volta battuto il tasto, il programma riprende dalla linea successiva. Aggiungiamo al programma precedente la seguente linea:

```
100 PRINT A$;: GOTO 10
```

Se ora digitiamo RUN, di può notare che sullo schermo non compare il cursore (■), ma qualsiasi carattere digitato viene visualizzato. Queste due linee di programma possono essere riscritte in un programma di editor di schermo, come mostrato in seguito.
Con l'editor di schermo si possono fare molte altre cose. Si può realizzare un cursore lampeggiante. Si possono gestire numerosi tasti, quali CLR/HOME , che provvedono ad eseguire la pulizia dello schermo. Si possono usare tasti di funzioni per rappresentare intere parole o frasi. A proposito di tasti funzione, le linee del seguente programma attribuiscono a ciascun tasto funzione uno scopo speciale. Questo è solo l'inizio di un programma che si può adattare alle nostre necessità.

```
20 IF A$ = CHR$(133) THEN POKE 53280,8:GOTO 10
30 IF A$ = CHR$(134) THEN POKE 53281,4:GOTO 10
40 IF A$ = CHR$(135) THEN A$= "GENTILE SIGNORE: " +CHR$(13)
```

```
50 IF A$ = CHR$(136) THEN A$="CORDIALI SALUTI, "+CHR$(13)
```

I numeri tra parentesi dei CHR$ provengono dalla tabella dei codici di CHR$ riportati nell'Appendice C; tale tabella riporta un numero differente per ciascun carattere. I quattro tasti funzionali sono impostati per eseguire i compiti rappresentati dalle istruzioni che seguono la parola THEN di ciascuna linea. Cambiando i numeri dentro le parentesi di CHR$, si possono designare caratteri differenti. Cambiando le informazioni dopo l'istruzione THEN, si possono eseguire istruzioni diverse.

COMPATTAZIONE DEI PROGRAMMI BASIC

Nei programmi BASIC possono essere compattate piu' istruzioni, rendendo il programma stesso piu' corto e piu' veloce possibile. Questo processo di riduzione dei programmi si chiama "compattazione".

La compattazione dei programmi permette di ridurre il numero massimo delle istruzioni di un programma. Con questo metodo si riduce molto anche la misura del programma, tanto da riuscire a farlo "girare" in uno spazio di memoria diversamente non sufficiente; senza contare che l'aumento di spazio che ne deriva consente la memorizzazione di una maggiore quantita' di dati.

ABBREVIAZIONE DELLE PAROLE CHIAVE

L'Appendice A riporta una lista delle abbreviazioni delle parole chiave. Cio' permette di inserire in una linea una maggiore quantita' di istruzioni. L'abbreviazione piu' comunemente usata e' il punto interrogativo (?), che sostituisce il comando PRINT. Tuttavia, richiedendo la lista di un programma con l'istruzione LIST, il COMMODORE 64 non ripete le abbreviazioni, ma visualizza le parole chiave per esteso. Se una linea di programma eccede gli 80 caratteri (2 linee dello schermo) con le parole chiave non abbreviate, e si desidera modificarla, e' necessario riscrivere tutta la linea con le parole chiave abbreviate, dopodiche' si puo' salvare il programma. Il salvataggio di un programma include le parole-chiave senza dilatare le linee, in quanto il COMMODORE 64 codifica con un singolo carattere le parole chiave del BASIC. In genere, le abbreviazioni vengono introdotte dopo che un programma e' stato scritto e non c'e' piu' bisogno di LISTarlo prima di salvarlo.

RIDUZIONE DEI NUMERI DI LINEA DI UN PROGRAMMA

Gran parte dei programmatori inizia i programmi dalla linea 100, assegnando ad ogni linea successiva un intervallo di 10 (ad esempio, 100, 110, 120). Cio' consente di inserire altre linee (111, 112, ecc.) man mano che il programma viene sviluppato.

ISTRUZIONI MULTIPLE SU UNA LINEA

Su ciascuna linea numerata del programma puo' trovare posto piu' di un'istruzione, ognuna separata dalle altre da due punti (:). L'unica

limitazione riguarda la lunghezza della linea, che non deve superare gli 80 caratteri compresi i due punti. Il seguente è un esempio di programma, prima e dopo la copattazione:

PRIMA DELLA COMPATTAZIONE

```
10 PRINT "HELLO. . .";
20 FOR T=1 TO 500:NEXT
30 PRINT "HELLO, AGAIN . . ."
40 GOTO 10
```

DOPO LA COMPATTAZIONE

```
10 PRINT "HELLO . . .";:FORT=1TO
   500:NEXT:PRINT"HELLO,
   AGAIN . . .":GOTO10
```

RIMOZIONE DELLE ISTRUZIONI REM

Le istruzioni REM sono utili per ricordarsi - o illustrare ad altri programmatori - quello che fa una certa parte del programma. Tuttavia, quando il programma e' completato nella sua versione definitiva, non c'e' piu' bisogno di tali istruzioni, per cui si puo' risparmiare un po' di spazio di memoria rimuovendole. Se in seguito si pensa di rivedere o studiare la struttura del programma, e' consigliabile tenerne una copia, completa di tutte le istruzioni REM.

VARIABILI

Se in un programma si usano ripetutamente un numero, una parola o una frase, e' utile, in genere, definire al loro posto una variabile che li contenga. Parole e frasi possono essere definite come variabili stringa, mediante l'uso di una lettera e del simbolo $. Un esempio puo' essere il seguente:

PRIMA DELLA COMPATTAZIONE

```
10 POKE 54296,15
20 POKE 54276,33
30 POKE 54273,10
40 POKE 54273,40
50 POKE 54273,70
60 POKE 54296,0
```

DOPO LA COMPATTAZIONE

```
10 V=54296:F=54273
20 POKEV,15:POKE54276,33
30 POKEF,10:POKEF,40:POKEF,70
40 POKEV,0
```

ISTRUZIONI READ E DATA

Grosse quantità di dati possono essere digitate una sola volta come un unico blocco di dati, ricominciando dall'inizio tutte le volte oppure, si può scrivere UNA SOLA VOLTA la parte di programma relativa alle istruzioni, e immettere questi valori ripetitivi in una o più linee di DATA.

VARIABILI CON INDICE E MATRICI

Le varibili con indice e matrici sono simili alle istruzioni DATA, poiché consentono il trattamento di grosse quantità di dati sotto forma di lista, dal quale la parte di programma relativa al trattamento dei dati preleva seguenzialmente questi ultimi. Le variabili con indice differiscono per la possibilità di avere tale lista a più dimensioni.

ELIMINAZIONE DEGLI SPAZI

Uno dei modi più facili per ridurre la misura di un programma è l'eliminazione degli spazi. Anche se spesso si introducono spazi nei programmi di prova per una maggiore chiarezza, in effetti non ce n'è alcun bisogno, e la loro eliminazione può comportare un risparmio di memoria.

PROCEDURE GOSUB

Se si fa un uso ripetuto di una particolare linea o istruzione, e' consigliabile saltare a quella linea da diversi punti del programma facendo ricorso all'istruzione GOSUB, anziche' scrivere tutta la linea ogni volta che se ne ha bisogno

TAB E SPC

Piuttosto che stampare diversi comandi di cursore per posizionare un carattere sullo schermo, spesso e' piu' economico usare le istruzioni TAB e SPC.

CAPITOLO 2
vocabolario del linguaggio BASIC

- Introduzione
- Parole Chiave. Abbreviazioni e Tipi di Funzione del BASIC
- Descrizione delle Parole Chiave del BASIC
- Tastiera e Caratteristiche del COMMODORE 64
- Editor di schermo

INTRODUZIONE

Questo capitolo spiega le parole chiave del linguaggio BASIC CBM. Per prima cosa viene riportato un prontuario delle parole chiave, che contiene anche le relative abbreviazioni e la rappresentazione sullo schermo di ciascuna lettera. Successivamente, vengono spiegati la sintassi ed il funzionamento di ciascuna parola chiave, riportando esempi esplicativi, del loro uso nei programmi.
Per convenzione, il BASIC del COMMODORE 64 permette di abbreviare la maggior parte della parole chiave. Le abbreviazioni sono introdotte nella macchina digitando il numero di lettere della parola chiave sufficiente per distinguerla dalle altre parole chiave; l'ultima lettera o carattere grafico viene immesso premendo il tasto SHIFT .
Quando vengono usate nel programma, le abbreviazioni NON risparmiano memoria, perché l'interprete BASIC riduce tutte le parole chiave ad un "token" di un singolo carattere. Un programma contenente delle abbreviazioni viene listato con le parole chiave nella loro forma intera. Le abbreviazioni possono essere usate per inserire più istruzioni in una linea di programma, anche se superano la lunghezza della linea logica di 80 caratteri dello schermo. L'editor di schermo lavora su una linea di 80 caratteri. Ciò significa che, se si usano abbreviazioni su qualsiasi linea che supera gli 80 caratteri, non siamo in grado di visualizzare questa linea al momento di listarla. Ciò che invece si deve fare è (1) digitare nuovamente l'intera linea, includendo tutte le abbreviazini, oppure (2) dividere i codici in due linee, ciascuna delle quali ha il proprio numero di linea, ecc.
Una lista completa di parole chiave, abbreviazioni, e della loro visualizzazione sullo schermo è presentata nella tabella 2.1. La lista è seguita da una descrizione di tutte le istruzioni, i comandi e le funzioni disponibili sul COMMODORE 64.
Questo capitolo mostra inoltre le funzioni interne dell'interprete del linguaggio BASIC. Le funzioni di sistema possono essere usate direttamente nelle istruzioni o in qualsiasi programma, senza dover definire prima la funzione stessa. Le stesse regole non valgono per le funzioni definite dall'utente. I risultati delle funzioni di sistema del BASIC possono essere usati come output immediati o possono essere assegnati a nomi di variabili di tipo appropriato.
Ci sono due tipi di funzioni BASIC:

1) NUMERICHE

2) STRINGA

Gli argomenti delle funzioni di sistema sono spesso racchiusi tra parentesi ( ). Le parentesi vengono poste dopo la parola chiave della funzione e NON SONO AMMESSI spazi tra l'ultima lettera della parola chiave e la parentesi di sinistra.
Il tipo di argomento necessario è generalmente deciso dal tipo di dato del risultato. Le funzioni che hanno come risultato un valore stringa sono identificate dalla presenza del segno dollaro ($) come ultimo carattere della parola chiave. In certi casi, le funzioni stringa contengono uno o più argomenti numerici.
Le funzioni numeriche convertono, quando necessario, tra formato intero e reale. Nella descrizione che segue, vengono mostrati i nomi delle funzioni con i relativi tipi di dato come valore di ritorno. I tipi degli argomenti vengono dati con il formato dell'istruzione.

PAROLE CHIAVE, ABBREVIAZIONI, TIPI DI FUNZIONE DEL BASIC

Tabella 2.1 - DESCRIZIONE DELLE PAROLE CHIAVE DEL BASIC

COMANDO	ABBR.	SCHERMO	FUNZIONE	COMANDO	ABBR.	SCHERMO	FUNZIONE
ABS	A SHIFT B	A	numerica	LET	L SHIFT E	L	
AND	A SHIFT N	A		LIST	L SHIFT I	L	
ASC	A SHIFT S	A ♥	numerica	LOAD	L SHIFT O	L	
ATN	A SHIFT T	A	numerica	LOG		LOG	numerica
CHR$	C SHIFT H	C	stringa	MID$	M SHIFT I	M	stringa
CLOSE	CL SHIFT O	CL		NEW		NEW	
CLR	C SHIFT L	C		NEXT	N SHIFT E	N	
CMD	C SHIFT M	C		NOT	N SHIFT O	N	
CONT	C SHIFT O	C		ON		ON	
COS		COS	numerica	OPEN	O SHIFT P	O	
DATA	D SHIFT A	D ♠		OR		OR	
DEF	D SHIFT E	D		PEEK	P SHIFT E	P	numerica
DIM	D SHIFT I	D		POKE	P SHIFT O	P	
END	E SHIFT N	E		POS		POS	numerica
EXP	E SHIFT X	E ♣	numerica	PRINT	?	?	
FN		FN		PRINT#	P SHIFT R	P	
FOR	F SHIFT O	F		READ	R SHIFT E	R	
FRE	F SHIFT R	F	numerica	REM		REM	
GET	G SHIFT E	G		RESTORE	RE SHIFT S	RE ♥	
GET#		GET#		RETURN	RE SHIFT T	RE	
GOSUB	GO SHIFT S	GO ♥		RIGHT$	R SHIFT I	R	stringa
GOTO	G SHIFT O	G		RND	R SHIFT N	R	numerica
IF		IF		RUN	R SHIFT U	R	
INPUT		INPUT		SAVE	S SHIFT A	S ♠	
INPUT#	I SHIFT N	I		SGN	S SHIFT G	S	numerica
INT		INT	numerica	SIN	S SHIFT I	S	numerica
LEFT$	LE SHIFT F	LE	stringa	SPC(	S SHIFT P	S	
LEN		LEN	numerica	SQR	S SHIFT Q	S ●	numerica

COMANDO	ABBR.	SCHERMO	FUNZIONE
STATUS	ST	ST	numerica
STEP	ST SHIFT E	ST	
STOP	S SHIFT T	S	
STR$	ST SHIFT R	ST	stringa
SYS	S SHIFT Y	S	
TAB(	T SHIFT A	T ♠	stringa
TAN		TAN	numerica
THEN	T SHIFT H	T	
TIME	TI	TI	numerica
TIME$	TI$	TI$	stringa
TO		TO	
USR	U SHIFT S	U ♥	numerica
VAL	V SHIFT A	V ♠	numerica
VERIFY	V SHIFT E	V	
WAIT	W SHIFT A	W ♠	

DESCRIZIONE DELLE PAROLE CHIAVE DEL BASIC

ABS

TIPO: Funzione Numerica
FORMATO: ABS(<espressione>)

Azione: Restituisce il valore assoluto del numero, che è il valore del numero senza il segno. Il valore assoluto di un numero negativo è il numero stesso moltiplicato per -1.

ESEMPI di Funzione ABS:

```
10 X = ABS ( Y )
10 PRINT ABS ( X * J )
10 IF X = ABS (X) THEN PRINT "POSITIVE"
```

AND

TIPO: Operatore
FORMATO: <espressione> AND <espressione>

Azione: La AND viene usata nelle operazioni Booleane per testare i Bit. Puo' essere usata anche per testare se entrambi gli operatori sono veri. Nell'algebra Booleana, il risultato della AND e' 1 solamente se entrambi gli operatori confrontati sono uguali a 1. Il risultato e'0 se uno solo o entrambi gli operatori sono uguale a 0 (falso).

ESEMPI dell'operazione AND su 1 Bit:

```
     0         1         0         1
 AND 0     AND 0     AND 1     AND 1
 -----     -----     -----     -----
     0         0         0         1
```

Il COMMODORE 64 esegue l'operazione AND su numeri compresi fra -32768 e +32767. Non si possono usare valori frazionari; i numeri al di fuori dell'intervallo causano un messaggio d'errore **?ILLEGAL QUANTITY** Quando viene convertito nel formato binario, questo numero viene trattato come numero a 16 bit. Bit corrispondenti vengono confrontati insieme, formando un risultato a 16 bit ancora compreso nell'intervallo di cui sopra.

ESEMPI di operazione AND su 16 Bit:

```
                          17
                      AND 194
             ----------------
             0000000000010001
       AND   0000000011000010
-----------------------------
(BINARIO)    0000000000000000
-----------------------------
(DECIMALE)                  0
```

```
                    32007
                AND 28761
          ----------------
          0111110100000111
     AND  0111000001011001
--------------------------
(BINARIO) 0111000000000001
--------------------------
(DECIMALE)          28673
```

```
                     -241
                AND 15359
          ----------------
          1111111100001111
     AND  0011101111111111
--------------------------
(BINARIO) 0011101100001111
(DECIMALE)          15119
```

Quando valuta se un numero e' vero o falso, il computer assume che tale numero sia vero purche' il suo valore non sia 0. Quando esegue un confronto, assegna il valore -1 se il risultato e' vero, 0 se il risultato e' falso. Quando valuta vero/falso con AND, ottiene vero come risultato solamente se ogni bit del risultato e'vero, cioè a 1.

ESEMPI dell'uso della AND nelle valutazioni VERO/FALSO:

```
50 IF X=7 AND W=3 THEN GOTO 10: REM VERO SOLO SE SONO VERI X=7 E W=3

60 IF A AND Q=7 THEN GOTO 10: REM VERO SE A <> 0 E Q=7
```

ASC

TIPO: Funzione Numerica
FORMATO: ASC (<stringa>)

Azione: ASC restituisce un numero da 0 a 255 corrispondente al valore ASCII Commodore del primo carattere della stringa. La tabella dei valori ASCII Commodore è riportata nell'appendice C.

ESEMPI della funzione ASC:

```
10 PRINT ASC("Z")
20 X = ASC("ZEBRA")
30 J = ASC(J$)
```

Se nella stringa non è presente alcun carattere, viene vusualizzato il messaggio d'errore **?ILLEGAL QUANTITY** Nel terzo esempio sopra, se J$=" " la funzione ASC da l'errore di cui sopra. Le istruzioni GET e GET# leggono un CHR$0 come stringa nulla.

Per eliminare questo inconveniente bisogna aggiungere CHR$(0) alla fine della stringa, come mostra l'esempio seguente:

ESEMPIO di funzione ASC che elimina l'errore ?ILLEGAL QUANTITY:

```
30 J = ASC(J$ + CHR$(0))
```

ATN

TIPO: Funzione Stringa
FORMATO: ATN (<numero>)

Azione: Questa funzione restituisce il valore dell'arcotangente di <numero>. Il risultato rappresenta l'angolo (in radianti) la cui tangente e'il numero dato. Il risultato e' sempre compreso fra /2 e +/2.

ESEMPI di funzione ATN:

```
10 PRINT ATN ( 0 )
20 X = ATN ( J ) * 180 / π        :REM CONVERSIONE IN GRADI
```

CHR$

TIPO: Funzione Stringa
FORMATO: CHR$ (<numero>)

Azione: Questa istruzione converte un codice ASCII Commodore nel suo carattere equivalente. Per la lista dei codici e dei caratteri equivalenti si veda l'Appendice C . Il <numero> deve essere compreso tra 0 e 255, altrimenti compare il messaggio **?ILLEGAL QUANTITY**

ESEMPI di funzione CHR$:

```
10 PRINT CHR$(65) : REM 65 =        A MAIUSCOLA

20 A$ = CHR$(13) : REM 13 =        TASTO RETURN
50 A = ASC(A$) : A$ = CHR$(A): REM        CONVERTE IN C64 ASCII, E VICEVERSA
```

CLOSE

TIPO: Istruzione di I/O
FORMATO: CLOSE (<numero di file>)

Azione: Questa istruzione chiude ogni collegamento tra qualsiasi file di dati o canale e un dispositivo. Il numero del file è lo stesso di quello riportato nell'istruzione di apertura del file o del dispositivo (OPEN) (vedere l'istruzione OPEN e la sezione sulla programmazione dell'input e dell'output). Quando si lavora con dispositivi di memorizzazione come registratori a cassetta ed unità disco, l'operazione CLOSE memorizza sulla cassetta o disco i

caratteri che sono rimasti nel buffer. Quando ciò non avviene, il file risulta incompleto su nastro e non leggibile su disco.
Per altri dispositivi, CLOSE non è altrettanto necessaria. Per ulteriori informazioni si veda il manuale dei dispositivi esterni.

ESEMPI di istruzioni CLOSE:

```
10 CLOSE 1
20 CLOSE X
30 CLOSE 9 * ( 1 + J )
```

CLR

TIPO: Istruzione
FORMATO: CLR

Azione: Questa istruzione rende disponibile la memoria RAM usata, normalmente per le variabili. Questa istruzione non influenza i programmi BASIC attualmente in memoria, bensì tutte le variabili, i puntatori dei GOSUB, dei cicli FOR....NEXT, le funzioni definite dall'utente ed i file, che vengono cancellati dalla memoria, rendendo così disponibile questo spazio per l'inserimento di nuove variabili, ecc.
Nel caso di file su nastro e su disco, l'istruzione CLR non esegue correttamente una CLOSE per quei file. Le informazioni riguardanti sia i file che i buffer non scaricati sono perse. Per ulteriori informazioni si veda l'istruzione CLOSE.

ESEMPI di istruzione CLR:

```
10 X=25
20 CLR
30 PRINT X

RUN
0

READY
```

CMD

TIPO: Istruzione di I/O
FORMATO: CMD <numero di file> [, <stringa>]

Azione: Questa istruzione trasferisce l'output dallo schermo TV alla periferica il cui file è specificato su disco, su nastro, su stampante o su un dispositivo di I/O come il modem. Il numero del file deve essere specificato nella precedente istruzione OPEN. La stringa, se riportata, viene inviata al file. Questa è utile per stampare listati, ecc.

Quando viene eseguito questo comando, qualsiasi istruzione PRINT e comando LIST non visualizza il testo sullo schermo, ma lo invia alla periferica con lo stesso formato. Per riportare di nuovo l'output sullo schermo, il comando PRINT# (numero di file) invia una linea vuota e disabilita CMD, in modo da disabilitare questo dispositivo dall'attesa di altri dati (dispositivo di #).

Qualsiasi errore di sistema (come **?SYNTAX ERROR**) viene visualizzato sullo schermo. Questo non pone i dispositivi nella condizione di non-ricezione, per cui si può inviare una linea vuota dopo una condizione d'errore (per ulteriori dettagli si veda il manuale della stampante e dell'unità disco).

ESEMPI dell'istruzione CMD:

```
OPEN 4, 4: CMD 4, "TITLE" :      LIST:REM LISTA IL PROGRAMMA SU STAMPANTE
PRINT# 4: CLOSE 4: REM          PONE LA STAMPANTE NELLA CONDIZIONE DI NON-
                                RICEZIONE, CHIUDENDO IL FILE AD ESSA DIRETTO
```

```
10 OPEN 1, 1, 1, "TEST":     REM CREA UN FILE SEQUENZIALE
20 CMD 1:    REM OUTPUT SU NASTRO ANZICHE' SU VIDEO
30 FOR L = 1 TO 100    100
40 PRINT L:     REM INSERISCE IL NUMERO NEL BUFFER DEL NASTRO
50 NEXT
60 PRINT# 1:      REM NON-RICEZIONE
70 CLOSE 1:       REM SCRIVE IL BUFFER NON COMPLETO, TERMINA CORRETTAMENTE
```

CONT

TIPO: Comando
FORMATO: CONT

Azione: Questo comando pone di nuovo in esecuzione un programma interrotto da STOP, da END o dalla pressione del tasto RUN/STOP Il programma riparte dal punto esatto dal quale era stato interrotto.

Mentre il programma è fermo, l'utente può controllare e cambiare qualsiasi variabile. Durante l'esame o la correzione del programma, l'istruzione STOP può essere sistemata in punti strategici per permettere di esaminare le variabili e di testare il flusso del programma Dall'editazione del programma risulta il messaggio d'errore **CAN'T CONTINUE** (anche se si è appena premuto messaggio d'errore RETURN con il cursore posizionato su una linea non modificata), se il programma si è fermato a causa di un errore interno o dell'Utente avvenuto prima di digitare CONT per far ripartire il programma.

ESEMPI dell'istruzione CONT:

```
10 PI=0:C=1
20 PI=PI+4/C-4/(C+2)
30 PRINT PI
40 C=C+4:GOTO 20
```

Questo programma calcola il valore di PI. Se si prova a lanciare (RUN) il programma e poco dopo si preme il tasto RUN/STOP , viene visualizzato:

BREAK IN 20 NOTA: potrebbe comparire un qualsiasi altro numero)

Per visualizzare il valore di C si può usare il comando PRINT C. Quindi si può usare CONT per ripartire da dove il COMMODORE 64 si era fermato.

COS

TIPO: Funzione
FORMATO: COS (<espressione numerica>)

Azione: - Questa funzione matematica calcola il coseno del numero, dove il numero rappresenta un angolo in radianti.

ESEMPIO di funzione COS:

```
10 PRINT COS ( 0 )
20 X = COS (Y * π / 180)     :REM CONVERTE I GRADI IN RADIANTI
```

DATA

TIPO: Istruzione
FORMATO: DATA (<lista di costanti>)

Azione: L'istruzione DATA memorizza informazioni all'interno di un programma. Il programma usa le informazioni per mezzo dell'istruzione READ, che preleva le costanti successive delle istruzioni DATA.
Le istruzioni DATA non devono essere eseguite dal programma ma devono essere solamente presenti. Di solito, perciò, sono situate alla fine del programma.
Tutte le istruzioni DATA di un programma vengono trattate come liste continue. Il dato è letto da sinistra a destra e dalla linea con numero inferiore fino alla linea con numero maggiore. Se l'istruzione READ incontra un dato che non risponde al tipo richiesto (se ad esempio vuole un numero ed incontra una stringa) si causa un messaggio d'errore.
Qualsiasi carattere può essere trattato come dato, ma talvolta il contenuto del dato deve essere racchiuso tra i doppi apici (" ") -caratteri come la virgola (,), i due punti (:), gli spazi vuoti, le lettere ottenute tenendo premuto il tasto SHIFT, la grafica e i caratteri di controllo cursore devono essere racchiusi tra i doppi apici.

ESEMPI di istruzione DATA:

```
10 DATA 1, 10, 5, 8
20 DATA JOHN, PAUL, GEORGE, RINGO
30 DATA "CARA MARY, COME STAI, AMORE, BILLY"
40 DATA -1.7E-9, 3.33
```

DEF FN

TIPO: Istruzione
FORMATO: DEF FN <nome> (<variabile>) = <espressione>

Azione: Questa istruzione imposta una funzione definita dall'utente utilizzabile successivamente nel programma. La funzione può essere costituita da qualsiasi formula matematica. Le funzioni definite dall'utente consentono di risparmiare spazio in quei programmi dove una lunga formula viene usata numerose volte. La formula necessita di essere specificata una sola volta, nell'istruzione di definizione; successivamente, viene abbreviata come nome di funzione. DEF FN deve essere dichiarata una sola volta; tutte le esecuzioni successive sono ignorate.

Il nome della funzione è costituito dalle due lettere FN seguite dal nome di una variabile, lungo uno o due caratteri, di cui il primo deve essere una lettera ed il secondo una lettera o un numero.

ESEMPI di istruzione DEF FN:

```
10 DEF FN A (X) = X + 7
20 DEF FN AA (X) = Y * Z
30 DEF FNA9 (Q) = INT( RND( 1)* Q+ 1)
```

La funzione viene richiamata piu' avanti nel programma usando il nome della funzione. Quest'ultimo viene usato come qualsiasi altra variabile; il suo valore viene calcolato automaticamente.

ESEMPI dell'uso di FN:

```
40 PRINT FN A (9)
50 R = FNAA (9)
60 G = G + FN A9 (10)
```

Nella linea 50, il numero 9 dentro le parentesi non influenza il risultato della funzione, perche' la funzione definita nella linea 20 non usa la variabile fra parentesi. Il risultato e' Y volte Z, senza tener conto del valore di Z. Nelle altre funzioni fra parentesi, invece, il numero 9 influenza il risultato.

DIM

TIPO: Istruzione
FORMATO: DIM <variabile> (<sottoscritti>), [<variabile>(<sottoscritti>).....]

Azione: Questa istruzione definisce un vettore o una matrice di variabili. Ciò permette di usare il nome della variabile con un indice; quest'ultimo punta all'elemento che sta per essere usato. Il numero più basso in un vettore è zero, il più alto è il numero riportato nell'istruzione DIM, che ha un massimo di 32767.

L'istruzione DIM deve essere eseguita una sola volta per ciascun

vettore, altrimenti si verifica l'errore **REDIM'D ARRAY** . Perciò, la maggior parte dei programmi esegue tutte le istruzioni DIM all'inizio. Il numero delle dimensioni può essere qualunque; gli indici del vettore sono al massimo 255. Tutto è limitato solamente dalla RAM disponibile per le variabili. I vettori possono essere costituiti da normali variabili numeriche, da stringhe o da numeri interi. Se le variabili sono di tipo stringa si usa il segno "$" dopo il nome della variabile, se sono numeri interi si usa il segno "%". Se un vettore a cui si fa riferimento in un programma non è mai stato dimensionato, viene dimensionato automaticamente a 11.

ESEMPIO di istruzione DIM:

```
10 DIM A ( 100 )
20 DIM Z ( 5, 7), Y ( 3, 4, 5)
30 DIM Y7% ( Q )
40 DIM PH$ (1000)
50 F (4) =9: REM ESEGUE AUTOMATICAMENTE DIM F(10)
```

ESEMPIO di SEGNAPUNTI DEL CALCIO ottenuto usando DIM:

```
 10 DIM S(1,5), T$(1)
 20 INPUT "NOMI DELLE SQUADRE";T$(0),T$(1)
 30 FOR Q=1 TO 5: FOR T=0 TO 1
 40 PRINT T$(T),    "PUNTEGGIO PARZIALE"Q
 50 INPUT S(T,Q): S(T,0)= S(T,0)+ S(T,Q)
 60 NEXT T,Q
 70 PRINT CHR$(147)      "TABELLONE"
 80 PRINT "TEMPO"
 90 FOR Q=1 TO 5
100 PRINT TAB( Q*2 +9) Q;
110 NEXT: PRINT TAB(15)       "TOTALE"
120 FOR T=0 TO 1: PRINT T$(T);
130 FOR Q=1 TO 5
140 PRINT TAB(Q*2 +9) S(T,Q);
150 NEXT: PRINT TAB(15) S(T,0)
160 NEXT
```

COME CALCOLARE LA MEMORIA USUFRUITA DA DIM:

5 bytes per il nome del vettore
2 bytes per ciascuna dimensione
2 bytes/elemento per le variabili intere
5 bytes/elemento per le variabili normali numeriche
3 bytes/elemento per le variabili stringa
1 byte per ogni carattere contenuto in ogni elemento stringa

END

TIPO: Istruzione
FORMATO: END

Azione: Questa istruzione termina l'esecuzione di un programma e fa vusualizzare il messaggio READY restituendo il controllo all'Utente. All'interno di un programma ci possono essere diverse istruzioni END.

Mentre non è affatto necessario includere più istruzioni END, non è necessario invece che un programma termini con una di esse.
L'istruzione END è simile allo STOP. L'unica differenza è che STOP visualizza il messaggio BREAK IN LINE XX , mentre END visualizza solo READY. Entrambi le istruzioni permettono al Computer di riprendere l'esecuzione dopo che si è digitato il comando CONT.

ESEMPIO di istruzione END:

```
10 PRINT "VUOI ESEGUIRE QUESTO PROGRAMMA"
20 INPUT A$
30 IF A$ = "NO"THEN END
40 ARRESTO DEL PROGRAMMA
999 END
```

EXP

TIPO: Funzione Numerica
FORMATO: EXP (<numero>)

Azione: - Questa funzione matematica calcola il valore ottenuto dall'elevamento a potenza della costante e (=2,71828183) per il numero dato Un valore maggiore di 88,0296919 causa il messaggio d'errore: ?OVERFLOW

ESEMPIO di funzione EXP:

```
10 PRINT EXP (1)
20 X = Y * EXP (Z * Q)
```

FN

TIPO: Funzione Numerica
FORMATO: FN <nome> (<numero>)

Azione: - Questa funzione si riferisce alla formula specificata dal nome vista in precedenza. Il numero (se riportato) viene inserito al suo posto e la formula calcolata. Il risultato e' un valore numerico.
Questa funzione puo' essere usata in modo diretto, basta che sia stata eseguita l'istruzione di DEFinizione.
Se si esegue una FN prima dell'istruzione DEF che la definisce, si verifica il messaggio d'errore UNDEF'D FUNCTION

ESEMPIO di funzione FN (definita dall'Utente):

```
PRINT FN A ( Q )
1100 J = FN J (7) + FN J (9)
9990 IF FN B7 (I+1)= 6 THEN END
```

FOR...TO...[STEP...]

TIPO: Istruzione
FORMATO: FOR <variabile> = <inizio> TO <fine> [STEP <incremento>]

Azione: - Questa e' un'istruzione speciale del BASIC che permette di usare facilmente una variabile come contatore. Si devono specificare alcuni parametri: il nome della variabile reale, il suo valore d'inizio, il limite del conteggio, e l'incremento da usare durante ciascun ciclo.
Quello seguente e' un semplice programma BASIC che conta da 1 a 10, stampando ciascun numero e terminando quando l'elaborazione e' completa. Non usa alcuna istruzione FOR:

```
100 L = 1
110 PRINT L
120 L = L + 1
130 IF L <= 10 THEN 110
140 END
```

LO STESSO PROGRAMMA, USANDO L'ISTRUZIONE FOR, DIVENTA:

```
100 FOR L = 1 TO 10
110 PRINT L
120 NEXT L
130 END
```

Come si può vedere, il programma è più corto e permette di capire più facilmente l'uso dell'istruzione FOR.
Quando viene eseguita l'istruzione FOR, hanno luogo numerose operazioni. Nella <variabile> che viene usata come contatore viene sistemato il valore di <partenza>. Nell'esempio precedente la variabile L assume il valore 1.
Quando viene raggiunta l'istruzione NEXT, alla <variabile> viene sommato il valore dell'<incremento>. Se non è incluso lo STEP, si assume l'<incremento> uguale a +1. La prima volta, il programma precedente, in riferimento alla linea 120, aggiunge 1 a L; il nuovo valore di L è quindi 2.
A questo punto, il valore della <variabile> è confrontato con il <limite>. Se il <limite> non è stato ancora raggiunto, il programma passa alla linea successiva a quella dell'istruzione FOR. Nel nostro caso, il valore contenuto in L (2) è minore del limite (10), perciò il programma passa alla linea 110.
Nel caso in cui il valore della <variabile> superi il <limite>, il ciclo viene interrotto e il programma riprende dall'istruzione successiva a NEXT. Nel nostro caso, il valore di L raggiungerà 11 e allora, essendo superato il limite (10), il programma passa alla linea 130.
Quando il valore dell'<incremento> è positivo, la <variabile> deve diventare maggiore del <limite>, quando è negativo deve diventare minore del <limite>.

NOTA: Un Loop viene sempre eseguito almeno una volta

ESEMPI dell'istruzione FOR...TO...STEP:

```
100 FOR L = 100 TO 0 STEP -1
100 FOR L = PI TO 6*π STEP .01
100 FOR AA = 3 TO 3
```

FRE

TIPO: Funzione
FORMATO: FRE (<variabile>)

Azione: - Questa funzione informa sulla quantita' di memoria RAM disponibile per programmi e variabili. Se un programma tenta di usare piu' spazio di quello disponibile, si verifica l'errore **OUT OF MEMORY**
Il numero tra parentesi puo' essere qualsiasi valore, e non viene usato nel calcolo.

NOTA: Se il risultato di FRE e' negativo aggiungere 65536 al risultato per avere il numero di byte disponibili in memoria

ESEMPIO di funzione FRE:

```
PRINT FRE ( 0 )
10 X = ( FRE ( K ) - 1000 ) / 7
950 IF FRE ( 0 ) < 100 THEN PRINT      "NON E' SUFFICENTE"
```

NOTA: La seguente espressione riporta sempre la quantita' attuale di memoria RAM disponibile: PRINT FRE(0)-(FRE(0)<0)*65536

GET

TIPO: Istruzione
FORMATO: GET <lista di variabili>

Azione: Questa istruzione legge tutti i tasti digitati dall'Utente. Mentre l'utente sta digitando, i caratteri vengono memorizzati nel buffer della tastiera del COMMODORE 64. Il Buffer può contenere fino a 10 caratteri; qualsiasi altro carattere digitato oltre il decimo viene perso. Via via che GET legge un carattere, si libera il posto per un altro carattere.
Se l'istruzione GET specifica dati numerici, e l'Utente degita tasti diversi da numeri, compare il messaggio d'errore **?SYNTAX ERROR** . Per essere sicuri, conviene leggere i tasti come stringhe e convertirli poi in numeri.
L'istruzione GET può essere usata per evitare alcuni limiti all'istruzione INPUT. Per ulteriori informazioni, vedere la sezione sull'uso dell'istruzione GET nella parte relativa alle Tecniche di Programmazione.

ESEMPI di istruzione GET:

```
10 GET A$: IF A$ = "" THEN 10: REM      RESTA FERMO A QUESTA ISTRUZIONE  FINCHE'
   HIT                                  NON SI BATTE UN TASTO
20 GET A$, B$, C$, D$, E$:   REM LEGGE 5 TASTI
30 GET A, A$
```

GET

TIPO: Istruzione
FORMATO: GET # <numero file>, <lista variabili>

Azione: Questa istruzione legge caratteri, uno alla volta, da un dispositivo o file specifico. Funziona come l'istruzione GET, solo che i dati provengono da dispositivi diversi dalla tastiera. Se non viene ricevuto alcun carattere, la variabile è impostata a stringa vuota (uguale a " ") o a 0 per variabili numeriche. I caratteri usati per separare i dati nei file, come la virgola (,) o il codice del tasto RETURN (codice ASCII=13), vengono ricevuti come qualsiasi altro carattere.

Quando GET# viene usata con il dispositivo #3 (schermo TV) essa legge un carattere alla volta dallo schermo. Ciascuna istruzione GET# sposta verso destra il cursore di 1 posizione. Il carattere posto alla fine della linea logica è cambiato in CHR$(13), cioè al codice del tasto RETURN.

ESEMPI dell'istruzione GET#

```
5 GET# 1, A$
10 OPEN 1, 3: GET# 1, Z7$
20 GET# 1, A, B, C$, D$
```

GOSUB

TIPO: Istruzione
FORMATO: GOSUB <numero di linea>

Azione: Questa è una forma particolare dell'istruzione GOTO, che implica una differenza importante: GOSUB si ricorda dove deve ritornare il controllo del programma. Quando nel programma viene trovata l'istruzione RETURN (differente dal tasto RETURN della tastiera), il controllo ritorna alla linea immediantamente successiva all'istruzione GOSUB di origine.

Il maggior uso di una sotto-procedura (GOSUB in realtà significa vai alla sotto-procedura) si ha quando un piccolo segmento di programma viene usata da diversi altri segmenti del programma. Usando le sotto-procedure, piuttosto che ripetere le stesse linee in diversi posti del programma, si risparmia molto spazio di memoria. In questo caso GOSUB è simile a DEF FN. DEF FN permette di salvare lo spazio usando una formula, mentre GOSUB salva spazio usando una routine di molte linee. Questo è un programma inefficiente che non usa la GOSUB:

```
100 PRINT "QUESTO PROGRAMMA SCRIVE"
110 FOR L = 1 TO 500 : NEXT
120 PRINT "LENTAMENTE SULLO SCHERMO"
130 FOR L = 1 TO 500 : NEXT
140 PRINT "USANDO UN SEMPLICE LOOP"
150 FOR L = 1 TO 500 : NEXT
160 PRINT "COME RITARDO DI TEMPO"
170 FOR L = 1 TO 500 : NEXT
```

Lo stesso programma che usa la GOSUB diventa:

```
100 PRINT"QUESTO PROGRAMMA SCRIVE"
110 GOSUB 200
120 PRINT "LENTAMENTE SUL VIDEO"
130 GOSUB 200
140 PRINT "USANDO UN SEMPLICE LOOP"
150 GOSUB 200
160 PRINT"COME RITARDO DI TEMPO"
170 GOSUB 200
180 END
200 FOR L = 1 TO 500 : NEXT
210 RETURN
```

Ogni volta che il programma esegue la GOSUB, sia il numero di linea che la posizione nel programma vengono salvati in un'area speciale detta "STACK" (pila) che occupa 256 bytes della memoria. Questo limita il numero di salti alle subroutines.
Perciò, il numero di indirizzi di ritorno della sottoprocedura che possono essere memorizzati è limitato, e si deve fare attenzione che ciascuna GOSUB abbia la relativa RETURN, perché in caso contrario si ha una scorretta uscita dalla subroutine con tutte le conseguenze che può portare.

GOTO

TIPO: Istruzione
FORMATO: GOTO <numero di linea> o GO TO <numero di linea>

Azione: - Questa istruzione permette al programma BASIC di eseguire le linee fuori dall'ordine numerico. La parola GOTO, seguita da un numero, fa saltare il programma alla linea indicata da quel numero. Una GOTO senza numero di linea equivale ad GOTO 0. Dopo l'istruzione GOTO ci deve sempre essere un numero di linea. Con GOTO e' possibile creare dei cicli che non hanno mai fine. Il piu' semplice esempio di quanto detto e' una linea che rimanda a se stessa, come 10 GOTO 10. Questi tipi di cicli possono essere fermarti usando il tasto RUN/STOP da tastiera.

ESEMPI d'istruzione GOTO

```
GOTO 100
10 GO TO 50
20 GOTO 999
```

IF...THEN...

TIPO: Istruzione
FORMATO: IF <espressione> THEN <numero di linea>
IF <espressione> THEN <istruzione>
IF <esprezzione> GOTO <numero di linea>

Azione: Questa è l'istruzione che da al linguaggio BASIC l'abilità di valutare condizioni e di compiere azioni diverse in base al risultato. La parola IF è seguita da un'espressione che può includere variabili, stringhe, numeri, confronti e operatori logici. La parola THEN compare nella stessa linea ed è seguita dal numero di linea oppure da una o più istruzioni BASIC. Quando l'espressione risulta falsa, tutto ciò che si trova dopo la parola THEN su quella linea viene ignorato e l'esecuzione continua con la successiva linea di programma. Se il risultato è vero, il programma salta al numero di linea dopo la parola THEN, oppure esegue qualsiasi altra istruzione BASIC che trova su quella linea.

ESEMPI d'istruzione IF...GOTO...

```
100 INPUT "BATTI UN NUMERO";N
110 IF N <= 0 GOTO 200
120 PRINT"RADICE QUADRATA=":SQR(N)
130 GOTO 100
200 PRINT"IL NUMERO DEVE ESSERE>0"
210 GOTO 100
```

Questo programma stampa la radice quadrata di qualsiasi numero positivo. L'istruzione IF è qui usata per l'INPUT. Quando il risultato di N 0 è vero, il programma salta alla linea 200, quando è falso sarà eseguita la linea 120. Si noti che THEN..GOTO..non è necessaria come IF..THEN, come si vede nella linea 100 dove GOTO 200 significa THEN..GOTO200.

ESEMPIO dell'istruzione IF...THEN...

```
100 FOR L = 1 TO 100
110 IF RND(1) < .5 THEN X = X+ 1 : GOTO 130
120 Y = Y+ 1
130 NEXT L
140 PRINT "IN TESTA E'"X
150 PRINT "IN CODA E'"Y
```

IF nella linea 110 controlla un numero casuale per vedere se e' < di 5. Quando il risultato e' vero, viene eseguita l'intera serie di istruzioni che seguono la parola THEN: per prima cosa X viene incrementata di 1, quindi il programma salta alla linea 130. Se il risultato e' falso, il programma passa all'istruzione successiva sulla linea 120.

INPUT

TIPO: Istruzione
FORMATO: INPUT [«<descrizione>] <lista di variabili>

Azione: Questa istruzione permette al programmatore di immettere dati nel computer. Quando viene eseguita, questa istruzione stampa un punto interrogativo (?) sullo schermo, e posiziona il cursore uno spazio a destra del punto interrogativo. A questo punto il Computer aspetta, Facendo lampeggiare il cursore, che l'operatore

digiti la risposta e prema il tasto RETURN .

Il comando INPUT può essere seguito da qualsiasi testo compreso fra i doppi apici (" "). Questo testo viene stampato sullo schermo e alla sua destra compare un punto interrogativo.

Dopo il testo, viene inserito un punto e virgola (;) e il nome di una o più variabili separate da virgola (,). Nella variabile verrà memorizzata l'informazione digitata dall'operatore. La variabile può essere qualsiasi nome legale di variabile, per cui si possono avere numerosi nomi di variabili, uno per ciascun INPUT diverso.

ESEMPI dell'istruzione INPUT:

```
100 INPUT A
110 INPUT B, C, D
120 INPUT"DESCRIZIONE";E
```

Quando questo programma "gira", appare un punto interrogativo per avvertire l'operatore che il COMMODORE 64 sta aspettando un inserimento di dati relativo alla variabile A. Qualsiasi numero digitato viene immobilizzato in A, per poter essere utilizzato nel programma. Se la risposta digitata non è un numero, appare il messaggio **?REDO FROM START** . significando che una stringa è stata ricevuta quando invece ci si aspettava un numero. Se l'operatore batte RETURN senza aver digitato nulla, il valore della variabile non cambia.

A questo punto compare un ? relativo all'INPUT della linea 110. Se digitiamo solamente un numero e battiamo RETURN il COMMODORE 64 visualizza due ??, il cui significato è la richiesta di più INPUT. Si possono digitare tanti INPUT quanti necessari, separati da virgole, che prevengono l'apparizione del doppio ??. Se vengono digitati più dati di quanti ne richiede l'istruzione INPUT, compare il messaggio **?EXTRA IGNORED** che significa che le voci digitate in più non vengono messe in alcuna variabile.

La linea 120 visualizza la parola DESCRIZIONE prima che compaia il ?. Il punto e virgola è richiesto tra la descrizione e la lista di variabili. L'istruzione INPUT non può mai essere usata fuori da un programma.

INPUT#

TIPO: Istruzione di I/O
FORMATO: INPUT# <numero di file>, <lista di variabili>

Azione: Questo è il metodo più veloce e più facile per recuperare i dati memorizzati su un file su disco o su nastro. Il dato è nella forma di variabili intera lunga massimo 80 caratteri; questa forma è opposta al metodo di un carattere alla volta della GET#. Per prima cosa, il file deve essere aperto, poi INPUT# può riempire le variabili.

Per ogni comando INPUT# una variabile è terminata quando legge un codice RETURN (CHR$(13)), una virgola (,), un punto e virgola (;) o due punti (:). Se necessario, durante la scrittura, si possono usare le virgolette (" ") per racchiudere questi caratteri (si veda l'istruzione PRINT#). Se la variabile è di tipo numerico e viene ricevuto un dato

non numerico, si verifica l'errore **BAD DATA** Con la INPUT# si possono leggere stringhe lunghe fino a 80 caratteri oltre i quali compare l'errore **STRING TOO LONG**
Quando questa istruzione viene usata con il dispositivo #3 (lo schermo), viene letta un'intera linea logica e spostato il cursore sulla linea successiva.

ESEMPI d'istruzione INPUT#

```
10 INPUT# 1, A
20 INPUT# 2, A$, B$
```

INT

TIPO: Funzione Intera
FORMATO: INT (<espressione numerica>)

Azione: Ritorna il valore intero di un'espressione. Se l'espressione è positiva la parte frazionaria, viene perduta, se l'espressione è negativa, la parte intera viene arrotondata all'intero-inferiore.

ESEMPI di funzione INT:

```
120 PRINT INT(99.4343), INT(-12.34)
 99      -13
```

LEFT$

TIPO: Funzione Stringa
FORMATO: LEFT$ (<stringa>,<N=deve essere un numero intero>)

Azione: Ritorna una stringa comprendente gli N caratteri più a sinistra della <stringa>. Il valore dell'argomento N deve essere compreso in un intervallo fra 0 e 255. Se N è maggiore della lunghezza della stringa, il risultato di ritorno è l'intera stringa. Se si usa N=0, ritorna una stringa nulla (cioè di lunghezza zero).

ESEMPI della funzione LEFT$:

```
10 A$ = "COMMODORE COMPUTERS"
20 B$ = LEFT$(A$,9): PRINT B$
RUN

COMMODORE
```

LEN

TIPO: Funzione Intera
FORMATO: LEN (<stringa>)

Azione: Restituisce il numero dei caratteri dell'espressione stringa. Vengono contati anche i caratteri non stampati e i blank.

```
CC$ = "COMMODORE COMPUTER": PRINT LEN(CC$)
 18
```

LET

TIPO: Istruzione
FORMATO: [LET] <variabile> = <espressione>

Azione: - L'istruzione LET puo' essere usata per assegnare un valore ad una variabile. Ma la parola LET e' opzionale, percio' i programmatori piu' esperti lasciano fuori LET perche' e' sottointesa e occupa memoria. Il segno di uguale (=) e' sufficiente quando si assegna il valore di un espressione al nome di una variabile.

ESEMPI d'istruzione LET:

```
10 LET D= 12       (e' lo stesso di D=12)
20 LET E$ = "ABC"
30 F$ = "PAROLE"
40 SUM$ = E$ + F$ (SUM$ e' uguale a ABCPAROLE)
```

LIST

TIPO: Comando
FORMATO: LIST [(<prima linea>) - (<ultima linea>)]

Azione: - Il comando LIST permette di visualizzare le linee del programma BASIC che si trova attualmente nella memoria del COMMODORE 64. Cio' consente di sfruttare la potenza dell'editor di schermo del computer per editare, velocemente e facilmente i programmi listati. Il comando di sistema LIST visualizza tutto o parte del programma che si trova attualmente in memoria sul dispositivo standard di OUTPUT. La LIST e' normalmente diretta allo schermo, si puo' usare CMD per riportare l'OUTPUT su un dispositivo esterno quale la stampante o il disco. Il Comando LIST puo' essere inserito in un programma, ma il BASIC ritorna al messaggio di sistema READY, dopo che una LIST e' stata eseguita. Quando si lista un programma sullo schermo, lo "scrolling" dal basso in alto puo' essere rallentato premendo il tasto di controllo CTRL. Digitando il tasto RUN/STOP si provoca l'interruzione dell'effetto di LIST.
Se nel comando di LIST non viene riportato alcun numero, sul video viene listato l'intero programma. Se viene specificato il numero di linea, seguito dal segno meno (-), vengono listate tutte le linee esistenti da quel numero di linea in poi. Se, invece, viene

specificato il numero, preceduto dal (-), allora vengono listate tutte le linee dall'inizio del programma fino a quella indicata dal numero stesso. Se sono indicati entrambi i numeri vengono listate le linee comprese nell'intervallo specificato dai due numeri di linea.

ESEMPI di comandi LIST:

```
LIST          lista tutto il programma in memoria
LIST 500      lista tutta la linea 500
LIST 150-     lista tutte le linee dalla 150 fino alla fine
LIST-1000     lista tutte le linee fino alla 1000
LIST150-1000  lista dalla linea 150 alla 1000 inclusa
```

```
10 PRINT "QUESTA E' LA LINEA 10"
20 LIST   LIST usata in MODO PROGRAMMA
30 PRINT "QUESTA E' LA LINEA 30
```

LOAD

TIPO: Comando
FORMATO: LOAD [«<nome del file>»] [, <canale>] [, <indirizzo>]

Azione: L'istruzione LOAD legge il contenuto su nastro o su disco e lo trasferisce in memoria. Si può così usare l'informazione caricata o cambiarla. Il numero della periferica è opzionale, ma se esso viene omesso il computer assume per default il numero 1, cioè il registratore. L'unità disco porta normalmente il numero 8. LOAD chiude tutti i file aperti e, se usata in modo diretto, esegue un CLR (azzeramento) prima di leggere il programma. Se la LOAD viene eseguita da programma, il programma viene subito fatto "girare". Ciò significa che si può usare la LOAD per "concatenare" più programmi assieme. Nessuna delle variabili viene azzerata durante l'operazione di concatenamento.

Se si usa un nome di file come mezzo di riconoscimento, il primo file corrisponde al nome usato viene caricato in memoria. Se stiamo lavorando con l'unità a dischi l'asterisco tra gli apici ("*") fa sì che venga caricato il primo programma della directory. Se il nome del file usato non è presente sul disco oppure non è il nome di un file di programma, compare il messaggio d'errore: **?FILE NOT FOUND** .

Quando si caricano programmi da nastro, il <nome del file>può essere omesso; in tal caso, viene caricato il primo file programma incontrato. Quando viene premuto il tasto PLAY, il COMMODORE 64 esegue un blank dello schermo che pertanto diventa di colore uniforme e senza alcuna scritta. Quando il file programma viene trovato lo schermo torna formato di partenza e viene visualizzato il messaggio FOUND.

Quando viene premuto il tasto C= , oppure dopo una pausa di 15 secondi, il file viene caricato. Se vine premuto il tasto SPACE BAR , il file del quale è stato trovato la testata viene saltato e si tenta di caricare il file successivo. I programmi sono caricati in memoria a partire dalla locazione 2048, a meno che non venga usato l'<indirizzo> indiretto secondario 1, nel qual caso il programma viene caricato nelle locazioni di memoria dalle quali è stato salvato.

ESEMPI del comando LOAD:

```
LOAD                  Carica il primo programma su nastro
LOAD A$               Usa il nome contenuto in A$ per cercare
LOAD"*",8             Carica il primo programma da disco
LOAD"",1,1            Cerca il primo programma su nastro e lo carica
                      nello stesso segmento di memoria di provenienza.

LOAD"STAR TREK"       Carica un file da nastro
PRESS PLAY ON TYPE
FOUND STAR TREK
LOADING
READY

LOAD"FUN",8           Carica un file da disco
SEARCHING FOR FUN
LOADING
READY

LOAD"GIOCO 1",8,1       Carica un file nella specifica locazione di
SEARCHING FOR GIOCO 1   memoria da cui il programma e' stato salvato
                        su disco.
LOADING
READY
```

LOG

TIPO: Funzione Reale
FORMATO: LOG (<numerica>)

Azione: - Ritorna il logaritmo naturale (logaritmo in base e) dell'argomento. Se il valore dell'argomento e' 0 o negativo il BASIC emette il messaggio d'errore: **?ILLEGAL QUANTITY**

ESEMPIO di funzione LOG:

```
25 PRINT LOG(45/7)
 1.86075234

10 NUM=LOG(ARG)/LOG(10) (calcola il LOGARITMO in base 10 di ALG)
```

MID$

TIPO: Funzione Stringa
FORMATO: MID$ (<stringa>, <espressione numerica-1>
[, <espressione numerica-2>)

Azione: La funzione MID$ restituisce una sottostringa prelevata dalla <stringa> argomento. La posizione di partenza della sottostringa è definita dall'argomento <espressione numerica 1>, la lunghezza della sottostringa dell'argomento <espressione numerica 2>.
Entrambi gli argomenti numerici possono avere valori compresi fra 0 e 255. Se l'<espressione 1> è maggiore della lunghezza della <stringa> o se l'<espressione 2> è zero, allora MID$ restituisce una stringa nulla. Se l<espressione 1> viene omessa, allora il computer assume come

lunghezza della sottostringa la differenza tra la lunghezza della stringa ed il valore dell'<espressione numerica 2>. Se la stringa sorgente è più corta dell'<espressione numerica 2>, allora viene usata l'intera stringa, dalla posizione di inizio fino alla fine.

ESEMPI di funzione MID$:

```
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$ + MID$(B$, 8, 8)
```

NEW

TIPO: Comando
FORMATO: NEW

Azione: Il comando NEW viene usato per cancellare il programma che si trova in quel momento nella memoria RAM del computer ed azzera tutte le variabili. NEW deve essere usato in modo diretto prima di digitare un nuovo programma, per azzerare la memoria. NEW può essere usata anche da programma, ma occorre tener presente che cancella tutto ciò che è stato fatto prima e che si trova ancora nella memoria del computer. Ciò può creare molti problemi durante la correzione di un programma.

ATTENZIONE: Non cancellare un vecchio programma prima di digitarne uno nuovo puo' causare confusione fra i due programmi.

ESEMPIO di comando NEW:

```
NEW      (Cancella il programma ed azzera le variabili)

10 NEW  (Esegue una NUOVA Operazione e ARRESTA il programma)
```

NEXT

TIPO: Istruzione
FORMATO: NEXT [<contatore>] [, <contatore>].....

Azione: L'istruzione NEXT viene usata con FOR per stabilire la fine di un ciclo FOR...NEXT. NEXT non deve essere posta necessariamente alla fine del gruppo di istruzioni appartenenti ala ciclo, ma è sempre l'ultima istruzione eseguita dal ciclo. Il <contatore> è il nome della variabile a indice del ciclo usata con FOR per dare inizio al ciclo. Una singola NEXT può arrestare numerosi cicli nidificati, quando sia seguita dai relativi nomi delle variabili <contatore> di ciascuna FOR. Per realizzare ciò, i nomi delle variabili devono comparire nell'ordine: dal ciclo più interno al ciclo più esterno. Quando si usa una singola NEXT per incrementare e fermare numerose variabili contatore, ciascun nome della variabile deve essere separato da una virgola. I cicli possono essere nidificati fino a 9 livelli. Se si omette il contatore della variabile (variabili), viene incrementato il contatore associato alla FOR del corrente livello di nidificazione. Quando viene raggiunta la NEXT, il valore del contatore viene

incrementato di 1 o del valore specificato dallo STEP opzionale. Viene quindi testato di nuovo il valore finale per vedere se il ciclo deve essere interrotto o meno. Un ciclo viene arrestato quando il valore del contatore, aggiornato con l'ultima NEXT, risulta maggiore del valore finale.

ESEMPIO di istruzione NEXT:

```
10 FOR J=1 TO 5: FOR K = 10 TO 20: FOR N = 5 TO -5 STEP -1
20 NEXT N, K, J     :REM      ARRESTA I CICLI NIDIFICATI

10 FOR L = 1 TO 100
20 FOR M = 1 TO 10
30 NEXT M
400 NEXT L                    SI NOTI COME I CICLI NON SI INCROCIANO

10 FOR A = 1 TO 10
20 FOR B = 1 TO 20
30 NEXT
40 NEXT                       NOTARE CHE NON E' NECESSARIO ALCUN NOME DI
VARIABILE
```

NOT

TIPO: Operatore Logico
FORMATO: NOT <espressione>

Azione: L'operatore logico NOT fa il "complemento" del valore di ciascun bit nel suo singolo operando, producendo come risultato intero il "complemento a 2". In altre parole, NOT significa "Se non è....". Quando si ha che fare con un numero reale, gli operandi vengono convertiti in interi e qualsiasi parte frazionaria viene perduta. L'operatore vero/falso, ottenuto come risultato di un test relazionale; quindi cambia il significato del confronto. Nel primo esempio seguente, se il "complemento a 2" di "AA" è uguale a "BB", e se "BB" NON è uguale a" "CC" allora l'espressione è vera.

ESEMPI dell'operatore NOT:

```
10 IF NOT AA = BB AND NOT(BB = CC) THEN . . . .

NN% = NOT 96: PRINT NN%
-97
```

NOTA: Per trovare il valore di NOT usare l'espressione x=(-(x+1)) (il complemento a 2 di qualsiasi intero si ottiene eseguendo il complemento al bit e sommando 1).

ON

TIPO: Istruzione
FORMATO: ON <variabile> GOTO/GOSUB <numero di linea> (, <numero di linea>).....

Azione: L'istruzione ON viene usata per saltare ad uno dei molti numeri di linea, di base al valore di una variabile compreso tra 0 e il numero delle linee date. Se il valore non è intero, la parte frazionaria viene persa. Per esempio, se la variabile ha valore 3, la ON esegue il GOTO al terzo numero di linea della lista. Se il valore della variabile è negativo compare il messaggio BASIC **?ILLEGAL QUANTITY** . Se il numero è zero o maggiore dei numeri delle linee riportate nella lista, il programma "ignora" l'istruzione e continua con l'istruzione che segue la ON.

La ON è in realtà una variante dell'istruzione IF...THEN. Invece di usare numerose istruzioni IF, ciascuna delle quali manda al programma specificato dalla linea, si usa la ON, che prende in considerazione una lista di numeri di linea a cui saltare. Osservando il primo esempio, si può notare che l'istruzione ON prende il posto di 4 istruzioni IF...THEN.

ESEMPI di istruzione ON:

```
ON -(A=7)-2*(A=3)- 3*(A<3)-4*(A>7)GOTO 400,900,1000,100
ON X GOTO 100,130,180,220
ON X+3 GOSUB 9000,20,9000
100 ON NUM GOTO 150, 300, 320, 390
500 ON SUM / 2 + 1 GOSUB 50, 80, 20
```

OPEN

TIPO: Istruzione di I/O
FORMATO: OPEN <numero del file>, (<dispositivo>) (, <indirizzo>) (, «<nome del file> (, <tipo>) (, <modo>)»)

Azione: - Questa istruzione apre un canale di INPUT/OUTPUT per un dispositivo periferico. Tuttavia non e' sempre necessario specificare tutte le parti dell'istruzione OPEN. Alcune istruzioni OPEN richiedono solo due parametri:

1) Numero del file logico
2) Numero del dispositivo o periferica

Il parametro <nome del file> è il numero del file logico, che pone in relazione le istruzioni OPEN, CLOSE, CMD, GET#, INPUT#, PRINT# le une con le altre, associandole al nome del file e al dispositivo che deve essere usato. Il numero del file logico deve essere compreso tra 1 e 255; si può usare liberamente uno di questi numeri.

NOTA: I numeri di file sopra 128 sono in realta'usati per altri scopi, per cui e' buona regola usare solamente i numeri fino a 127.

Ciascun dispositivo del sistema ha un proprio numero identificazione. Il numero di 〈dispositivo〉 e' usato nella OPEN per specificare su

quale dispositivo è presente il file. Per le periferiche quali cassette, unità a disco o stampanti, si richiedono anche i diversi indirizzi secondari. Questi ultimi possono essere immaginati così codici che dicono quale operazione deve eseguire ciascun dispositivo. Il numero del dispositivo del file logico viene usato con, ogni GET#, INPUT#, PRINT#. Se il numero del <dispositivo> o periferica viene omesso, il computer assume automaticamente le informazioni vengono inviate o ricevute dal Datasette(TM), che porta il numero 1. Anche il nome del file può essere omesso, ma nel programma non è poi possibile richiamare quel file se non gli è stato assegnato un nome nella OPEN. Quando si registrano file su registratori a cassetta, e si omette l'indirizzo secondario, allora il computer assume 0 come <indirizzo> secondario (operazione di lettura).

1 come <indirizzo> secondario apre il file su cassetta per un'operazione di scrittura. 2 come <indirizzo> secondario provoca la scrittura di un segnale di fine nastro alla successiva chiusura del file. L'indicatore di fine nastro protegge da una lettura accidentale oltre la fine dei dati, che provocherebbero il messaggio d'errore del BASIC ?DEVICE NOT PRESENT

Per i file su disco, sono disponibili gli indirizzi secondari da 2 a 14 per i file dati, mentre gli altri numeri hanno un significato speciale per i comandi DOS (Per ulteriori dettagli si veda il manuale del drive del disco con i comandi DOS).

Il (nome del file)e' una stringa da 1 a 16 caratteri ed e' opzionale per i file su cassetta e su stampante. Se viene omesso il (tipo) di file, si assume per default un file programma, a meno che non sia specificato il (modo).

I file sequenziali vengono aperti per la lettura con (modo)=R, a meno che non sia stato specificato il (modo)=W, implicando cosi' che il file deve essere aperto per la scrittura. Il (tipo) di file puo' essere usato per aprire un file relativo esistente. Usare REL per (tipo) con i file relativi, che, insieme a quelli sequenziali, si trovano solamente su disco.

Se si tenta di accedere ad un file prima di averlo aperto, si verifica il messaggio d'errore ?FILE NOT OPEN . Se si apre un file su disco per la scrittura e questo file esiste gia', compare il messaggio d'errore FILE EXISTS . Non ci sono test di questo tipo per i file su nastro, per cui si deve essere sicuri che il nastro sia posizionato esattamente, altrimenti si puo' scrivere sopra dati precedentemente salvati Se si apre un file gia' aperto compare il messaggio BASIC: FILE OPEN (per ulteriori dettagli vedere il manuale per la stampante).

ESEMPI di istruzione OPEN:

```
10 OPEN 2,8,4"DISK-OUTPUT SEQ,W" (Apre un file sequenziale su disco)

10 OPEN 1,1,2"TAPE-WRITE' (Scrive un carattere EOF, cioè fine di un file,
                           alla chiusura del file)
10 OPEN 50,0 (Input da tastiera)

10 OPEN 12,3 (Output da video)

10 OPEN 1,1,0,"NOME" (Legge da cassetta)

10 OPEN 1,1,1,"NOME" (Scrive su cassetta)

10 OPEN 1,2,0,CHR$(10) (Apertura di un canale su periferica RS-232)
```

10 OPEN 1,4,0,"STRINGA (Invia alla stampante il set maiuscolo/grafica)

10 OPEN 1,4,7,"STRINGA' (Invia alla stampante il set maiuscolo/minuscolo)

10 OPEN 1,5,0,"STRINGA' (Invia alla stampanta il set maiuscolo/grafica. Stampante che sia come periferica N. 5)

OR

TIPO: Operatore Logico
FORMATO: <operando> OR <operando>

Azione: - Come gli operatori relazionali possono essere usati per prendere delle decisioni riguardanti il flusso del programma, cosi' gli operatori logici possono collegare due o piu' relazioni e dare come risultato il valore vero/falso, che puo' poi essere usato in una decisione. Quando viene usato in un calcolo, l'operatore OR da' come risultato 1 se i corrispondenti bit di uno o di tutti e due gli operandi sono uguali a 1. Il risultato di questo confronto e' un numero intero, il cui valore dipende da quello degli operandi. Quando invece viene usato in un confronto, l'operatore OR ha la capacita' di unire due espressioni in una singola espressione composta. Se una delle due espresioni, o entrambi, risulta vera, il risultato dell'espressione composta e' vero (-1). Nel primo degli esempi che seguono, se AA e' uguale a BB OPPURE XX e' 20 allora l'espressione e' vera.
Gli operatori logici funzionano convertendo gli operandi in numeri interi a 16 bit, con segno, rappresentati in complemento a 2 e compresi nell'intervallo -32768. . +32767. Se gli operandi non sono compresi in questo intervallo, si genera un messaggio di errore. Ciascun bit del risultato viene determinato a partire dal valore dei corrispondenti bit dei due operandi.

ESEMPI dell'operatore OR.

100 IF (AA = BB) OR (XX = 20) THEN . . . . !

230 KK% = 64 OR 32: PRINT KK%
(Quest'istruzione e' stata battuta assumendo un valore di 1000000 per 64, ed un valore di 100000 per 32)

96 (Il computer ha risposto con il valore binario 1100000 = 96)

PEEK

TIPO: Funzione Intera
FORMATO: PEEK (<espressione numerica>)

Azione: Restituisce un valore intero compreso nell'intervallo 0...255, letto da una locazione di memoria. L'<espressione numerica> è una locazione di memoria che deve essere compresa fra 0 e 65535, altrimenti il BASIC

emette il messaggio di errore **?ILLEGAL QUANTITY**

ESEMPI di funzione PEEK:

10 PRINT PEEK(53280) AND 15	(Ritorna il valore del colore di bordo dello schermo)
5 A%=PEEK(45)+PEEK(46)*256	(Ritorna un indirizzo riportato nella tabella delle variabili del BASIC)

POKE

TIPO: Istruzione
FORMATO: POKE <locazione>, <valore>

Azione: - L'istruzione POKE viene usata per scrivere il valore binario di un byte (8 bit) in una locazione di memoria o in un registro di input/output. La <locazione> è un'espressione aritmetica il cui risultato deve essere compreso fra 0 e 65535; il <valore> è fra 0 e 255. Se un valore supera i limiti di questi due intervalli, compare il messaggio BASIC **?ILLEGAL QUANTITY**
Le istruzioni PEEK e POKE (funzioni interne che considerano una locazione di memoria) sono utili per memorizzare dati, per controllare la grafica o per generare suoni, per caricare sottoprocedure in linguaggio assembly e per passare argomenti e risultati a/da una sottoprocedura in linguaggio assembly. Inoltre, usando l'istruzione PEEK si possono esaminare i parametri del Sistema Operativo, mentre l'uso dell'istruzione POKE consente il loro trattamento e la loro modifica. Una mappa completa delle locazioni utili e' riportata in Appendice G.

ESEMPIO di istruzione POKE:

POKE 1024, 1	(Scrive una "A" nella posizione 1 dello schermo)
POKE 2040, PTR	(Aggiorna il puntatore dati dell'animazione 0)
10 POKE RED, 32	
20 POKE 36879, 8	
2050 POKE A, B	

POS

TIPO: Funzione Intera
FORMATO: POS (<dummy>)

Azione: - Riporta la posizione attuale del cursore, che e' compresa fra 0 (carattere piu' a sinistra) e 79 per una linea logica di schermo lunga 80 caratteri. Dato che il COMMODORE 64 ha un video con 40 colonne, qualsiasi posizione da 40 a 79 si riferisce alla seconda linea logica dello schermo. L'argomento dummy viene ignorato.

ESEMPIO di funzione POS:

```
1000 IF POS(0) > 38 THEN PRINT CHR$(13)
```

PRINT

TIPO: Istruzione
FORMATO: PRINT [<variabile>] [<, /;> <variabile>].....

Azione: - L'istruzione PRINT viene comunemente usata per scrivere dati sullo schermo. Tuttavia, si puo' ricorrere all'istruzione CMD per riindirizzare l'output a qualsiasi altro dispositivo del sistema. La <variabile> nella lista di output puo' essere un'espressione di qualunque tipo. Se non viene riportata alcuna lista di output, viene stampata una linea bianca. La posizione di ciascuna voce stampata e' determinata dalla punteggiatura usata per separare le voci riportate sulla lista di output.
I caratteri di punteggiatura utilizzabili sono gli spazi, le virgole o i punti e virgola. La linea logica di schermo di 80 caratteri e' divisa in otto zone di stampa di 10 spazi ciascuna. Se si introduce una virgola nella lista delle espressioni, la stampa del prossimo carattere inizia nella zona di stampa successiva. Il punto e virgola fa si' che il prossimo valore venga stampato immediatamente dopo il valore precedente. Tuttavia, ci sono due eccezioni a questa regola:

1) Gli elementi numerici sono seguiti da uno spazio aggiuntivo.
2) I numeri positivi sono preceduti da uno spazio.

Lo stesso effetto del punto e virgola puo' essere ottenuto usando, come separatori fra le costanti stringa o i nomi delle variabili, uno spazio o nessun tipo di punteggiatura. Va tuttavia notato che gli spazi tra una stringa ed un numero, o tra due numeri, interrompono l'output senza che il secondo elemento venga stampato.
Se alla fine della lista di output si pone una virgola o un punto e virgola, la prossima istruzione PRINT comincia la stampa sulla stessa linea; se invece alla fine della lista non compare alcuna puntaggiatura, dopo l'ultimo dato viene stampato un ritorno carrello ed un incremento di linea, per cui la prossima istruzione PRINT inizia la stampa sulla linea successiva. Se l'output destinato allo schermo e' piu' lungo di 40 colonne, viene continuato sulla linea successiva dello schermo
Non c'è in pratica comando BASIC che possa essere impiegato in tanti modi come <istruzione> PRINT. Associata a questa istruzione, infatti, si trova una tale varietà di simboli, funzioni e parametri, da far considerare questa istruzione come un linguaggio a sè stante interno al BASIC.

ESEMPI di istruzione PRINT:

1)
```
5 X = 5
10 PRINT -5*X, X-5, X+5, X↑5

-25        0         10        3125
```

2)
```
5 X=9
10 PRINT X;     "AL QUADRATO =";X*X;"E";
```

```
20 PRINT X ; "AL CUBO ="X 3

9 AL QUADRATO = 81 E 9 AL CUBO = 729
```

3)
```
90 AA$="ALPHA":BB$="BAKER": CC$="CHARLIE":DD$="DOG": EE$="ECHO"
100 PRINT AA$BB$;CC$ DD$,EE$

ALPHABAKERCHARLIEDOG        ECHO
```

1. MOVIMENTO DEL CURSORE

I controlli cursore programmabili con il "modo virgolette" sono:

TASTO	VISUALIZZAZIONE
CLR/HOME	S
SHIFT CLR/HOME	♥
⇑ CRSR ⇓	Q
SHIFT ⇑ CRSR ⇓	●
⇐CRSR⇒	]
SHIFT ⇐CRSR⇒	▌

Se ad esempio si vuole stampare in diagonale la parola HELLO, partendo dall'angolo in alto a sinistra, si deve digitare:

PRINT " CLR/HOME H ⇑ CRSR ⇓ E ⇑ CRSR ⇓ L ⇑ CRSR ⇓ L ⇑ CRSR ⇓ O"

che viene visualizzato:

PRINT " S H Q E Q L Q L Q O"

2. CARATTERI «REVERSE»

Premendo i tasti CTRL e 9 dentro le virgolette appare il carattere R, che fa si' che tutti i caratteri vengano stampati in VIDEO REVERSE (come il negativo di un disegno). Per terminare la stampa "reverse", premere CTRL e Ø, dopodiche' compare il carattere ⊟, oppure viene stampato un RETURN [CHR$(13)] (lo stesso risultato si ottiene chiudendo l'istruzione PRINT con una virgola o un punto e virgola).

3. CONTROLLI DEL COLORE

Premendo il tasto CTRL o il tasto C=, insieme ad uno degli otto tasti colore, compare dentro le virgolette un carattere "reverse"

speciale quando questo carattere viene stampato, si verifica il cambiamento di colore.

TASTO	COLORE	VISUALIZZAZIONE
CTRL 1	Nero	
CTRL 2	Bianco	
CTRL 3	Rosso	
CTRL 4	Azzurro	
CTRL 5	Porpora	
CTRL 6	Verde	
CTRL 7	Blu	
CTRL 8	Giallo	
C= 1	Arancio	
C= 2	Marrone	
C= 3	Rosso chiaro	
C= 4	Grigio 1	
C= 5	Grigio 2	
C= 6	Verde chiaro	
C= 7	Blu chiaro	
C= 8	Grigio 3	

Se si vuole stampare la parola HELLO in azzurro e la parola THERE in bianco, occorre digitare:

PRINT " [CTRL] [4] HELLO [CTRL] [2] THERE"

che viene visualalizzato:

PRINT " [▚] HELLO [E] THERE"

4. MODO INSERIMENTO

Gli spazi creati con il tasto [INST/DEL] hanno alcune caratteristiche comuni al modo virgolette. I controlli cursore ed i controlli colore appaiono come caratteri "reverse". L'unica differenza sono [INST] e [DEL] nel modo virgolette DEL esegue la sua normale funzione, mentre in questo caso visualizza la [T] Ed [INST] , che nel modo virgolette crea un carattere speciale, ora inserisce normalmente gli spazi.
Cio' perche' e' possibile creare un'istruzione PRINT contenente DEL , che nel modo virgolette non puo' essere stampato. Il modo di operare per ottenere quanto detto e' illustrato nel seguente esempio:

10 PRINT"HELLO" [INST/DEL] [SHIFT] [INST/DEL] [SHIFT] [INST/DEL] [INST/DEL] [INST/DEL] P"

che viene visualizzato:

10 PRINT"HELLO [T] [T] P"

Quando la precedente linea viene eseguita, la parola visualizzata e' HELP, poiche' le ultime due lettere sono cancellate e sostituite da P.

AVVERTENZA: DEL e' attiva sia con LIST che con PRINT, per cui e' difficile editare una linea con questi caratteri.

La condizione "modo inserimento" termina quando viene premuto il

tasto RETURN (o SHIFT RETURN), oppure quando vengono digitati tanti caratteri quanti sono gli spazi inseriti.

5. ALTRI CARATTERI SPECIALI

Ci sono altri caratteri che possono essere stampati per funzioni speciali, anche se non sono facilmente disponibili da tastiera. Per inserire questi caratteri fra virgolette, si possono lasciare sulla linea spazi a loro riservati, premendo RETURN o SHIFT RETURN , e ritornare agli spazi con il controllo cursore. Per iniziare a digitare i caratteri in "reverse", bisogna premere CTRL RVS/ON , e quindi i seguenti tasti:

FUNZIONE	TASTO	
SHIFT RETURN	SHIFT M	
Imposta le minuscole	N	
Imposta le maiuscole	SHIFT N	
Disabilita i tasti di impostazione	H	
Abilita i tasti di impostazione	I	

SHIFT RETURN funziona sia con PRINT che con LIST, per cui l'uso di questi tasti rende quasi impossibile l'editazione; anche la LISTa di un programma appare in un modo molto strano.

PRINT#

TIPO: Istruzione di I/O
FORMATO: PRINT# <numero-file> [<variabile] [<, /;> <variabile].....

Azione: L'istruzione PRINT# viene usata per scrivere dati su un file logico, per il quale deve essere adoperato lo stesso numero usato nell'istruzione OPEN relativa a quel file. L'uscita dei dati è indirizzata al numero di dispositivo usato nell'istruzione OPEN. L'espressione <variabile> della lista di output può essere di qualunque tipo. La punteggiatura tra gli elementi è la stessa dell'istruzione PRINT e può essere usata nella stessa maniera. Gli effetti della punteggiatura sono invece diversi per due importanti mitivi.
Quando si usa PRINT# con un file su nastro, la virgola, anziche' spaziare le zone di stampa, ha lo stesso effetto del punto e virgola; percio', se non si usano spazi, virgole, punti e virgola o altri tipi di punteggiatura per separare i dati, l'effetto sulla spaziatura e' lo stesso. I dati vengono scritti come un flusso continuo di caratteri Gli elementi numerici sono seguiti da uno spazio e, se positivi, sono preceduti da uno spazio.
Se la lista termina senza alcun carattere di punteggiatura, alla fine dei dati viene scritto un ritorno carrello e si avanza di una riga; se invece la lista termina con una virgola o un punto e virgola, il ritorno carrello e l'avanzamento della linea vengono soppressi. Senza badare alla punteggiatura, la successiva istruzione PRINT# inizia a stampare nella posizione del primo carattere disponibile. Quando viene usata l'istruzione INPUT#, l'avanzamento linea agisce come uno stop, lasciando la variabile vuota al momento dell'esecuzione della successiva INPUT#. L'avanzamento linea puo' essere soppresso o compensato come viene mostrato nel'esempio seguente.
Il modo piu' facile per scrivere piu' di una variabile su un file su

nastro o su disco e' quello di impostare la variabile stringa a CHR$(13), ed usare questa stringa come separatore delle variabili, al momento della scrittura del file.

ESEMPI di istruzione PRINT:

1)
```
10 OPEN 1,1,1,"TAPE FILE"
20 R$ = CHR$(13)
30 PRINT# 1,1;R$;2;R$;3;R$;4;R$;5
40 PRINT# 1,6
50 PRINT# 1,7
```

(Cambiando CHR$(13) in CHR$(44) si usa come separatore una virgola; se invece si usa CHR$(59), si ottiene un punto e virgola).

2)
```
10 CO$=CHR$(44): CR$=CHR$(13)
20 PRINT#1, "AAA"CO$"BBB", "CCC";"DDD";"EEE"CR$ "FFF"CR$;
30 INPUT#1, A$,BCDE$,F$
```

```
AAA,BBB        CCCDDDEEE
(ritorno carrello)
FFF (ritorno carrello)
```

3)
```
 5 CR$=CHR$(13)
10 PRINT#2, "AAA";CR$;"BBB"
20 PRINT#2, "CCC";
30 INPUT#2, A$,B$,DUMMY$,C$
```

```
(10 spazi) AAA
BBB
(10 spazi) CCC
```

READ

TIPO: Istruzione
FORMATO: READ <variabile> [,<variabile>].....

Azione: L'istruzione READ è usata per riempire i nomi di variabili con il contenuto delle costanti provenienti dalle istruzioni DATA. Il tipo di dato letto deve essere conforme al tipo di variabile specificata, altrimenti compare il messaggio BASIC **?SYNTAX ERROR** (*). Le variabili della lista di input dell'istruzione DATA devono essere separate da una virgola.
Una singola istruzione READ puo' accedere ad una o piu' istruzioni DATA, l'accesso alle quali avviene in ordine (vd. DATA), oppure diverse istruzioni READ possono accedere alla stessa istruzione DATA. Se vengono eseguite piu' READ di quanti siano gli elementi dell'istruzione DATA, allora compare il messaggio BASIC **?OUT OF DATA**.
Se il numero delle variabili specificate e' minore del numero di elementi dell'istruzione DATA, allora la successiva READ inizia a leggere dal successivo elemento della lista (vd. RESTORE)

NOTA: Il messaggio **?SYNTAX ERROR** al quale abbiamo segnato vicino un (*) riporta il numero di linea della istruzione DATA, e NON quello dell'istruzione READ.

ESEMPI di istruzione READ:

```
110 READ A,B,C$
120 DATA 1,2,HELLO

100 FOR X=1 TO 10: READ A(X):NEXT

200 DATA 3.08, 5.19, 3.12, 3.98, 4.24
210 DATA 5.08, 5.55, 4.00, 3.16, 3.37

1 READ CITY$,STATE$,ZIP
5 DATA DENVER,COLORADO,80211
```

(Riempie la lista degli elementi secondo l'ordine delle costanti (linea 5)).

REM

TIPO: Istruzione
FORMATO: REM [<commento>]

Azione: - L'istruzione REM permette di commentare un programma in modo tale da renderlo piu' comprensibile al momento di LISTarlo. Il commento rappresenta cio' che si vuole fare con un'istruzione o ungruppo di istruzioni in un particolare contesto del programma; ad esempio, puo' servire per ricordare che una variabile viene usata per un determinato scopo. Il commento puo' essere costituito da qualsiasi testo, parola o carattere, compresi anche i due punti (:) o le parole chiave del BASIC.
Tutto cio' che viene scritto dopo REM, e che sta sullo stesso numero di linea, viene ignorato dal BASIC, ma i commenti vengono regolarmente stampati nello stesso modo in cui sono stati immessi. Un'istruzione REM puo' essere riferita ad una GOTO o ad una GOSUB; l'esecuzione del programma continua con la prossima linea eseguibile del programma (tale linea deve avere numero maggiore della precedente, e contenere un'istruzione eseguibile).

ESEMPI di istruzione REM:

```
10 REM CALCOLO DELLA VELOCITA' MEDIA
20 FOR X=1 TO 20,  :REM CICLO DI VENTI VALORI
30 SUM=SUM + VEL(X): NEXT
40 AVG=SUM/20
```

RESTORE

TIPO: Istruzione
FORMATO: RESTORE

Azione: Il BASIC mantiene un puntatore interno ai prossimi DATA. Quindi, mano a mano che viene eseguito un READ sui DATA questo puntatore si incrementa. Se è necessario rileggere i contenuti dei DATA dall'inizio dovremo utilizzare il comando RESTORE che appunto riposiziona il detto puntatore.

ESEMPIO di istruzione RESTORE:

```
100 FOR X=1 TO 10: READ A(X): NEXT
200 RESTORE
300 FOR Y=1 TO 10: READ B(Y): NEXT

4000 DATA 3.08, 5.19, 3.12, 3.98, 4.24
4100 DATA 5.08, 5.55, 4.00, 3.16, 3.37
```

(Riempie le due matrici con dati identici)

```
10 DATA 1,2,3,4
20 DATA 5,6,7,8
30 FOR L=1 TO 8
40 READ A: PRINT A
50 NEXT
60 RESTORE
70 FOR L=1 TO 8
80 READ A: PRINT A
90 NEXT
```

RETURN

TIPO: Istruzione
FORMATO: RETURN

Azione: L'istruzione RETURN viene usata per uscire da una sottoprocedura alla quale eravamo andati con un GOSUB. Permette di continuare l'esecuzione delle istruzioni che si trovano dopo la GOSUB. Se si opera con sottoprocedure nidificate, ciascuna GOSUB deve essere accoppiata ad un'istruzione RETURN finale. Una sottoprocedura puo' contenere un numero qualsiasi di istruzioni RETURN, ma la prima di tali istruzioni che viene incontrata fa uscire dalla sottoprocedura.

ESEMPIO di istruzione RETURN:

```
10 PRINT"QUESTO E' IL PROGRAMMA"
20 GOSUB 1000)
30 PRINT"IL PROGRAMMA CONTINUA"
40 GOSUB 1000
50 PRINT"ANCORA PROGRAMMA"
60 END
1000 PRINT:"QUESTO E' IL GOSUB":RETURN
```

RIGHT$

TIPO: Funzione Stringa
FORMATO: RIGHT$ (<stringa>, <espressione numerica>)

Azione: - La funzione RIGHT$ restituisce una sottostringa presa dalla parte piu' a destra fino alla fine della <stringa> argomento. La lunghezza della sottostringa e' definita dall'argomento <espressione numerica>, che puo' essere un intero compreso fra 0 e 255. Se il valore dell'espressione numerica e' 0, viene ritornata una stringa

nulla (""); se tale valore e' maggiore della lunghezza della (stringa), viene ritornata l'intera stringa.

ESEMPIO di funzione RIGHT$:

```
10 MSG$ = "COMMODORE COMPUTERS"
20 PRINT RIGHT$(MSG$,9)
RUN

COMPUTERS
```

RND

TIPO: Funzione Reale
FORMATO: RND (<espressione numerica>)

Azione: La funzione RND crea un numero casuale (random) compreso fra 0.0 e 1.0. Il computer genera una sequenza di numeri casuali eseguendo dei calcoli su un numero di partenza chiamato SEED. Questo viene generato all'accensione del sistema. L'argomento <espressione numerica> è privo di significato, ad eccezione del segno (positivo, o negtivo) oppure a meno che sia zero.
Se l'argomento è negativo, viene ritornata la stessa sequenza di numeri pseudocasuali, a partire dal valore di SEED. SEED diversi ritornano sequenze numeriche diverse, ma ciascuna sequenza è ripetitiva, partendo infatti dallo stesso numero di seme. Per eseguire dei test sui programmi, fa comodo avere a disposizione una sequenza di numeri "casuali" conosciuta.
Se come argomento dell'<espressione numerica> si sceglie 0, RND genera un numero direttamente dal clock di sistema. Un argomento negativo implica che per ogni chiamata alla funzione RND viene dato un nuovo SEED.

ESEMPI di funzione RND:

```
220 PRINT INT(RND(0)*50)            (Ritorna un numero casuale compreso fra 0
                                     e 49)

100 X=INT(RND(1)*6)+INT(RND(1)*6)+2     (Simula due dadi)
100 X=INT(RND(1)*1000)+1
100 X=INT(RND(1)*150)+100           (Numeri casuali comresi fra 100 e 249)

100 X=RND(1)*(U-L)+L                (Numeri casuali compresi fra U, estremo
                                        superiore, e L, estremo inferiore)
```

RUN

TIPO: Comando
FORMATO: RUN [<numero di linea>]

Azione: - Il comando di sistema RUN e' usato per lanciare l'elaborazione del programma attualmente in memoria. Il comando RUN causa, prima della partenza del programma, l'esecuzione

dell'istruzione CLR; tuttavia, tale operazione di azzeramento puo' essere evitata usando CONT o GOTO, al posto di RUN, per rilanciare l'elaborazione. Se si specifica il <numero di linea>, il programma inizia da quella linea, altrimenti ha inizio dalla prima linea di programma. Il comando RUN puo' essere usato anche all'interno di un programma. Se il <numero di linea> specificato non esiste, compare il messaggio BASIC **UNDEF'D STATEMENT**
Quando si raggiunge una END, una STOP o l'ultima linea di programma, oppure se si verifica un errore durante l'esecuzione, il programma in corso di elaborazione si ferma, restituendo il controllo al BASIC, che si posiziona sul modo diretto.

ESEMPI di comando RUN:

RUN (Inizia dalla prima linea del programma)
RUN 500 (Inizia dalla linea 500)
RUN X (Inizia dalla linea X oppure causa UNDEF'D STATEMENT ERROR se la linea X non esiste)

SAVE

TIPO: Comando
FORMATO: SAVE [«<nome del file>»] [,<numero del dispositivo>] [,<indirizzo>]

Azione: Il comando SAVE viene usato per memorizzare su nastro o disco il programma attualmente in memoria. Durante il funzionamento di SAVE, il programma che deve essere salvato non può essere interessato da altri comandi. Il programma salvato rimane ancora in memoria, anche dopo il termine dell'operazione, finché non venga coperto da un altro programma o non si usi qualche comando, si ha cioè solo un'operazione di copia il tipo di file deve essere "PRG" (programma). Se si omette il <numero del dispositivo>, allora il COMMODORE 64 assume come dispositivo su cui salvare il programma il dispositivo numero 1, cioè il nastro. Se il <numero del dispositivo> è 8, allora il programma viene salvato su disco. L'istruzione SAVE può essere usata anche all'interno di un programma: al termine dell'operazione di salvataggio, l'esecuzione riprende dalla linea successiva alla SAVE.
I programmi su nastro sono automaticamente memorizzati due volte, in modo che il COMMODORE 64 possa eseguire dei controlli per scoprire eventuali errori, quando il programma viene nuovamente caricato in memoria. Quando un programma viene salvato su nastro, il <nome del file> e l'<indirizzo> secondario sono facoltativi, anche se il nome del programma tra gli apici (" ") o in una variabile stringa (---$) facilita all'Utente il compito della ricerca. Se un file viene salvato senza un nome, la successiva ricerca usando un nome non ha esito.
Un indirizzo secondario 1 ordina al sistema operativo di caricare il programma su nastro a partire da una locazione di memoria diversa da quella standard, cioè la locazione 2048. Un indirizzo secondario 2 fa seguire al programma un segnale di fine nastro. Un indirizzo secondario 3 esegue entrambe queste funzioni.
Quando si salva un file su disco, il nome del file deve essere sempre presente.

ESEMPI di comando SAVE:

```
SAVE              (Memorizza su nastro senza assegnare un nome al file)
SAVE "ALPHA", 1   (Memorizza su nastro con il nome "ALPHA")
SAVE "ALPHA", 1, 2 (Memorizza "ALPHA" seguito dall'indicatore di fine
                  nastro)
SAVE "FUN.DISK",8 (Salva su disco [dispositivo 8 = disco])
SAVE A$           (Memorizza su nastro con il nome contenuto in A$)
10 SAVE "HI"      (Salva il programma epassa alla linea successiva)
SAVE"ME",1,3      (Memorizza nella stessa locazione di memoria, ponendo
                  al termine un indicatore di fine nastro)
```

SGN

TIPO: Funzione Intera
FORMATO: SGN (<espressione numerica>)

Azione: - SGN ritorna un valore intero in base al segno dell'argomento <espressione numerica>: se tale argomento e' positivo il risultato e' 1, se e' negativo il risultato e' -1, se e' nullo il risultato e' 0.

ESEMPIO di funzione SGN:

```
90 ON SGN(DV)+2 GOTO 100, 200, 300
(Salta a 100 se DV=negativo, a 200 se DV=nullo, a 300 se DV=positivo)
```

SIN

TIPO: Funzione Reale
FORMATO: SIN (<espressione numerica>)

Azione: - SIN ritorna il valore del seno, espresso in radianti, dell'<espressione numerica>. Il valore di COS(X) e' uguale a SIN(X+3.14159265/2).

ESEMPIO di funzione SIN:

```
235 AA = SIN(1.5): PRINT AA
 .997494987
```

SPC

TIPO: Funzione Stringa
FORMATO: SPC (<espressione numerica>)

Azione: - La funzione SPC viene usata per controllare la formattazione dei dati sia sullo schermo che in un file logico. Il numero di spazi dati dall'argomento <espressione numerica> viene stampato a partire dalla prima posizione disponibile. Per lo schermo e per il nastro tale valore deve essere compreso tra 0 e 255, per il disco fino a 254. Per

i file su stampante, quest'ultima esegue un ritorno carrello ed un avanzamento linea nel caso che venga stampato uno spazio nell'ultima posizione carattere di una linea. Sulla linea seguente non viene stampato alcuno spazio.

NOTA: per le stampanti è bene comunque consultare gli appositi manuali.

ESEMPIO di funzione SPC:

```
10 PRINT "RIGHT "; "HERE &";
20 PRINT SPC(5) "OVER" SPC(14) "THERE"
RUN

RIGHT HERE &     OVER              THERE
```

SQR

TIPO: Funzione Reale
FORMATO: SQR (<espressione numerica>)

Azione: - SQR ritorna il valore della radice quadrata dell'argomento <espressione numerica>. Questo valore non deve essere negativo, altrimenti compare il messaggio BASIC **?IL LEGAL QUANTITY**.

ESEMPIO di funzione SQR:

```
FOR J = 2 TO 5: PRINT J*5, SQR(J * 5): NEXT

     10   3.16227766
     15   3.87298335
     20   4.47213595
     25   5
          READY
```

STATUS

TIPO: Funzione Intera
FORMATO: STATUS

Azione: Ritorna lo stato dell'ultima operazione di input/output eseguita su un file aperto. Lo stato può essere letto da qualsiasi dispositivo periferico. La parola chiave dello stato (ST) è il nome di una variabile definita dal sistema in cui il KERNAL inserisce lo stato delle operazioni di I/O. La seguente è la tabella dei valori dei codici di stato per operazioni su file su nastro, stampante, disco ed RS-232.

POSIZIONE DEL BIT ST	VALORE NUMERICO DI ST	VERIFICA NASTRO + CARICO	LETTURA REGISTRATORE	R/W DEL BUS SERIALE
0	1		Fuori tempo in scrittura	
1	2		Fuori tempo in scrittura	
2	4	Blocco corto		Blocco corto
3	8	Blocco lungo		Blocco lungo
4	16	Errore di lettura non recuperabile		Qualsiasi errore
5	32	Errore di "checksum"		Errore di "checksum"
6	64	Fine file	EOI	
7	-128	Fine nastro	Dispositivo non presente	Fine nastro

ESEMPI di funzione STATUS:

```
10 OPEN1,4:OPEN2,8,4,"MASTER FILE,SEQ,W"
20 GOSUB 100 : REM CONTROLLA LO STATO
30 INPUT#2,A$,B,C
40 IF STATUS AND 64 THEN 80 : REM TRATTAMENTO DELLA FINE DEL FILE
50 GOSUB 100 : REM CONTROLLA LO STATO
60 PRINT#1,A$,B;C
70 GOTO 20
80 CLOSE1:CLOSE2
90 GOSUB 100:END
100 IF ST > 0 THEN 9000 : REM TRATTAMENTO ERRORE DI I/O DEL FILE
110 RETURN
```

STEP

TIPO: Istruzione
FORMATO: [STEP <espressione>]

Azione: - La parola chiave opzionale STEP segue il <valore finale> che compare nell'istruzione FOR, definendo l'incremento della variabile contatore del ciclo. STEP consente di usare come incremento qualsiasi valore diverso da zero. Se viene omessa, il valore dell'incremento e' +1. Quando in un ciclo FOR si raggiunge l'istruzione NEXT, viene presa in considerazione STEP, quindi viene nuovamente testato il valore finale per controllaare se il ciclo e' terminato (per maggiori informazioni si veda l'istruzione FOR).

ESEMPI di istruzione STEP:

```
25 FOR XX = 2 TO 20 STEP 2        (Ripete il ciclo 10 volte)
35 FOR ZZ = 0 TO -20 STEP -2      (Ripete il ciclo 11 volte)
```

STOP

TIPO: Istruzione
FORMATO: STOP

Azione: - L'istruzione STOP viene usata per arrestare l'esecuzione del programma attuale e per ritornare al modo diretto. Digitando da tastiera RUN/STOP si ottiene lo stesso effetto dell'istruzione STOP, per cui il BASIC visualizza il messaggio d'errore **?BREAK IN LINE nnnnn** , seguito da READY (nnnnn = numero di linea dove si e' verificata l'interruzione). Qualsiasi file aperto viene mantenuto tale; tutte le variabili mantengono i rispettivi valori, permettendo cosi' di essere esaminate. Il programma puo' essere fatto ripartire usando le istruzioni CONT o GOTO.

ESEMPI di istruzione STOP:

```
10 INPUT#1, AA, BB, CC
20 IF AA = BB AND BB = CC THEN STOP
30 STOP
```

```
BREAK IN LINE 20 (Se AA vale -1 e BB=CC)
BREAK IN LINE 30 (In tutti gli altri casi)
```

STR$

TIPO: Funzione Stringa
FORMATO: STR$ (<espressione numerica>)

Azione: - Ritorna la rappresentazione stringa del valore numerico dato dall'argomento. Al momento della conversione del valore dell'<espressione numerica>, tutti i numeri sono seguiti da uno spazio e, se positivi, preceduti da uno spazio.

ESEMPIO di funzione STR$:

```
100 FLT = 1.5E4: ALPHA$ = STR$(FLT)
110 PRINT FLT, ALPHA$

 15000    15000
```

SYS

TIPO: Istruzione
FORMATO: SYS <locazione di memoria>

Azione: Questo è il modo più comune per fondere programmi BASIC con programmi in linguaggio macchina. Questi ultimi cominciano dalla locazione di memoria indicata dalla SYS. Il comando di sistema SYS viene usato sia in modo diretto che da programma per trasferire il controllo dal microprocessore ad un programma in linguaggio macchina esistente in memoria. La locazione di memoria data è un'espressione numerica e può essere qualunque locazione di memoria ROM o RAM.

Quando si usa l'istruzione SYS, la parte in linguaggio macchina deve terminare con l'istruzione RTS, in modo che, al termine del programma in linguaggio macchina, l'elaborazione venga ripresa dal BASIC con l'esecuzione dell'istruzione successiva alla SYS.

ESEMPI di istruzione SYS:

```
SYS 64738      (Salta alla ROM riservata alla Partenza a Freddo)
10 POKE 4400,96: SYS 4400      (Salta alla locazione 4400 del codice
                               macchina, e ritorna immediatamente)
```

TAB

TIPO: Funzione Stringa
FORMATO: TAB (<espressione numerica>)

Azione: - La funzione TAB sposta il cursore di un numero di spazi dato dall'argomento (espressione numerica), a partire dalla posizione piu' a sinistra della linea attuale. Il valore dell'argomento e' compreso fra 0 e 255. L'istruzione TAB puo' essere usata solo con l'istruzione PRINT, dato che non ha alcun effetto se usata con PRINT# per un file logico.

ESEMPIO di funzione TAB:

```
100 PRINT "NOME" TAB(25),  "QUANTITA'": PRINT
110 INPUT#1, NAM$, AMT$
120 PRINT NAM$ TAB(25) AMT$ AMT$

NAME                        QUANTITA'

G.T. JONES           25.
```

TAN

TIPO: Funzione Reale
FORMATO: TAN (<espressione numerica>)

Azione: - Ritorna il valore, in radianti, della tangente espressa dall'argomento (espressione numerica). Se per la funzione TAN si verifica un overflow, il BASIC visualizza il messaggio ?DIVISION BY ZERO

ESEMPIO di funzione TAN:

```
10 XX = .785398163: YY = TAN(XX): PRINT YY
1
```

TIME

TIPO: Funzione Numerica
FORMATO: TI

Azione: La funzione TI legge il contenuto dell'orologio interno. Questo tipo di "orologio" è chiamato "jiffy clock". All'accensione del sistema il "jiffy clock" viene impostato a zero (inizializzazione). Questo temporizzatore, come sarebbe più giusto chiamarlo, si incrementa ogni 1/60 di secondo e viene spento durante un I/0 su nastro.

ESEMPIO di funzione TI:

```
10 PRINT TI/60 "Secondi dall'accensione"
```

TI$

TIPO: Funzione Stringa
FORMATO: TI$

Azione: - Il timer TI$ funziona come un orologio reale per tutta la durata dell'accensione del sistema. L'intervallo dell'orologio (o "jiffy clock") letto viene usato per aggiornare il valore di TI$, usato successivamente come "stringa di tempo", formata da sei caratteri rappresentanti le ore, i minuti ed i secondi. Il timer TI$ puo' anche assumere un punto arbitrario di partenza, funzionando in questo caso come un cronometro. Dopo un I/O su nastro, il valore di TI$ non e' esatto.

ESEMPIO di funzione TI$:

```
1 TI$ = "000000": FOR J=1 TO 10000: NEXT: PRINT TI$

000011
```

USR

TIPO: Funzione Reale
FORMATO: USR (<espressione numerica>)

Azione: - La funzione USR salta ad una sottoprocedura scritta in linguaggio macchina dall'Utente, il cui indirizzo di partenza e' puntato dal contenuto delle locazioni 785-786. L'indirizzo di partenza viene stabilito, prima di chiamare la funzione USR, usando l'istruzione POKE per impostare le locazioni suddette; in caso contrario, si ha il messaggio di errore ?ILLEGAL QUANTITY.
Il valore dell'<espressione numerica> viene memorizzato nell'accumulatore reale a partire dalla locazione 97, per permettere l'accesso al codice in linguaggio macchina; il risultato della funzione USR è il valore contenuto in quella locazione quando la sottoprocedura fa ritorno al BASIC.

ESEMPI di istruzione USR:

```
10 B = T * SIN(Y)
20 C = USR (B/2)
30 D = USR (B/3)
```

VAL

TIPO: Funzione Numerica
FORMATO: VAL (<stringa>)

Azione: - Ritorna un valore numerico rappresentante il dato contenuto nell'argomento (stringa). Se il primo carattere della stringa diverso dallo spazio (" ") non e' il piu' (+), il meno (-) o un numero, il valore ritornato da VAL e' 0. La conversione della stringa termina quando viene incontrata la fine della stringa stessa, oppure al primo carattere diverso da una cifra (ad eccezione del punto decimale e della E esponenziale).

ESEMPIO di funzione VAL:

```
10 INPUT#1, NAM$, ZIP$
20 IF VAL(ZIP$) < 19400 OR VAL(ZIP$) > 96699
   THEN PRINT NAM$ TAB(25) "GREATER PHILADELPHIA"
```

VERIFY

TIPO: Comando
FORMATO: VERIFY [«<nome del file>»], [, <dispositivo>]

Azione: - Il comando VERIFY viene usato in modo diretto per confrontare il contenuto di un programma BASIC, registrato su nastro o su disco, con il programma attualmente in memoria. In genere viene usato subito dopo una SAVE, per essere sicuri della corretta memorizzazione del programma.
Se si omette il numero del <dispositivo> la verifica avviene sul registratore Datasette [TM]. Per i file su nastro, se si omette il <nome del file>, il programma attualmente in memoria viene confrontato con il programma trovato sul nastro. Per i file su disco, il <nome del file> deve essere presente. Una qualsiasi differenza riscontrata nel testo del programma causa la visualizzazione del messaggio BASIC ?VERIFY ERROR.
Il nome del programma puo' essere passato sia direttamente, all'interno degli apici (""), sia attraverso una variabile stringa. VERIFY e' anche usato per posizionare il nastro esattamente dopo la fine dell'ultimo programma, allo scopo di evitare errori accidentali di sovrascrittura, nelsalvataggiodiunnuovoprogramma.

ESEMPI di comando VERIFY:

```
VERIFY                    (Controlla il primo programma su nastro)
PRESS PLAY ON TAPE
OK
SEARCHING
FOUND <FILENAME>
VERIFYING

9000 SAVE "ME",8:
9010 VERIFY "ME",8        [Cerca il programma sul disopsitivo 8 (disco)]
```

WAIT

TIPO: Istruzione
FORMATO: WAIT <locazione>, <maschera 1> [, <maschera 2>]

Azione: - L'istruzione WAIT fa si' che l'esecuzione del programma venga sospesa fino al riconoscimento, da parte di un indirizzo di memoria, di una specificata configurazione di bit. In altre parole, WAIT puo' essere usata per arrestare un programma finche' non si verifichino alcuni eventi esterni. Cio' viene svolto controllando lo stato dei bit dei registri di I/O. Il dato usato con WAIT puo' essere

rappresentato da una qualunque espressione numerica, ma deve essere convertito in un valore intero.

Per la maggioranza dei programmatori, questa istruzione non dovrebbe essere mai usata: essa causa infatti l'arresto del programma finche' i bit di una locazione di memoria specificata non cambino in modo specifico; questo fatto puo' tornare utile solamente per determinate operazioni di input/output.

WAIT estrae il valore dalla locazione di memoria ed esegue un'operazione di AND logico con i valori della <maschera 1>. Se nell'istruzione viene riportata anche la <maschera 2>, allora viene eseguita una OR esclusiva tra il risultato della prima operazione e la <maschera 2>. In altri termini, la maschera 1 "esclude" qualsiasi bit non si desideri testare. Per ogni bit uguale a 0 nella maschera 1, il corrispondente bit del risultato e' 0. Il valore della maschera 2 interessa tutti i bit, in modo che il controllo possa avvenire sia per condizione attivata che per condizione disattivata. Tutti i bit testati per zero devono avere un 1 nella corrispondente posizione della maschera 2.

Se i bit corrispondenti degli operandi della <maschera 1> e della <maschera 2> differiscono, la OR esclusiva ha come risultato un bit a 1, altrimenti tale bit e' 0. Tramite l'istruzione WAIT si puo' creare una pausa di lunghezza infinita, nel qual caso il sistema viene ripristinato premendo i tasti RUN/STOP e RESTORE (premere RESTORE, mantenendo premuto RUN/STOP). Il primo esempio attende che si prema un tasto del registratore, prima di proseguire l'elaborazione del programma; il secondo esempio attende che un'animazione entri in contatto con lo sfondo dello schermo.

ESEMPI di istruzione WAIT:

```
WAIT 1,32,32
WAIT 53273,6,6
WAIT 36868,144,16     (144 e 16 sono maschere (144=10010000, 16=10000).
                       WAIT ferma il programma finche' il bit 128 non
                       sia in ON, o il bit 16 non sia in OFF)
```

TASTIERA DEL COMMODORE 64 E SUE CARATTERISTICHE

Il Sistema Operativo ha un buffer di tastiera di 10 caratteri, usato per contenere i dati digitati prima che vengano elaborati. Questo buffer, o coda, mantiene i caratteri dei tasti premuti nell'ordine di entrata (il primo tasto ad essere stato premuto genera il carattere elaborato per primo). Se ad esempio viene immesso un secondo carattere prima che il precedente sia stato elaborato, il secondo carattere viene memorizzato nel buffer mentre continua l'elaborazione del primo carattere. Al termine di tale elaborazione, il buffer della tastiera viene controllato per vedere se contiene altri caratteri, dando cosi' inizio all'elaborazione del carattere generato dal secondo tasto premuto. Senza questo buffer, un'immissione rapida di caratteri da tastiera potrebbe provocare perdite accidentali di caratteri.

In altre parole, il buffer della tastiera permette di "digitare in anticipo" sul sistema, abbreviando i tempi di risposta del sistema ai messaggi di INPUT o alle istruzioni GET. Durante la battitura, i valori dei rispettivi caratteri vengono disposti in coda nel buffer, cioè vengono messi uno dietro l'altro dove attendono l'elaborazione

nell'ordine di ingresso. Questa caratteristica può generare il problema occasionale delle battiture accidentali di un carattere, per cui il programma può prelevare dal buffer un carattere non corretto. carattere non corretto.
Normalmente, i dati non corretti immessi nel sistema non causano alcun problema, poiche' possono essere corretti con il tasto di spostamento a sinistra del cursore (⇐CRSR), oppure cancellati (tasto INST/DEL e ribattuti correttamente. Se invece e' gia' stato eseguito un ritorno carrello (e' gia' stato premuto il tasto RETURN), non e' più possibile alcuna correzione, poiché tutti i caratteri contenuti nel buffer e compreso il ritorno carrello vengono elaborati prima di qualsiasi correzione. Questa situazione può essere evitata usando un ciclo di caricamento del buffer della tastiera prima di leggere una risposta.

```
10 GET JUNK$:IF JUNK <> "" THEN 10:REM VUOTA IL BUFFER DELLA TASTIERA
```

Oltre che con GET e INPUT, l'ultimo tasto premuto può essere letto con PEEK, andando a prendere da una locazione di memoria (ad esempio, la 197 [$00CS HEX]) il valore del tasto appena battuto. Se al momento dell'esecuzione della PEEK non si è battuto alcun tasto, viene restituito il valore 64. I valori numerici ed i simboli della tastiera, insieme ai corrispondenti codici carattere (CHR$), sono riportati nell'Appendice C. Il seguente esempio mostra un ciclo mantenuto attivo fino alla prima battuta di un testo.

```
10 AA=PEEK(197):IF AA=64 THEN 10
20 BB$=CHR$(AA)
```

La tastiera puo' essere vista come un insieme di interruttori organizzato in una matrice di 8 righe per 8 colonne. Questa matrice viene eseguita dal KERNAL alla ricerca degli interruttori chiusi (e quindi dei tasti premuti) per mezzo del circuito di I/O CIA#1 (Adattatore Interfaccia Complessa MOS 6526). Per eseguire questa scansione si utilizzano due registri del CIA: il registro 0, allocato in 56320 ($DC00 HEX), per le colonne, ed il registro 1, allocato in 56321 ($DC01 HEX), per le righe.
I bit 0-7 della locazione di memoria 56320 corrispondono alle colonne 0-7, i bit 0-7 della locazione di memoria 56321 corrispondono alle righe 0-7. Il KERNAL decodifica gli interruttori impostati nei corrispondenti valori CHR$(n) dei tasti premuti, prima scrivendo in sequenza i valori delle colonne, poi leggendo i valori delle righe.
Il prodotto delle otto righe per le otto colonne da' 64 valori possibili; tuttavia, se il primo tasto tasto ad essere battuto e' RVS CTRL o C=, oppure SHIFT seguito da un carattere, allora si generano dei valori addizionali. Cio' perche' il KERNAL decodifica questi tasti separatamente, "ricordandosi" quando uno dei tasti di controllo e' stato premuto. Il risultato della scansione della tastiera e' poi memorizzato nella locazione 197.
I caratteri possono anche essere scritti direttamente nel buffer della tastiera, dalla locazione 631 alla 640, ricorrendo all'istruzione POKE. In questo caso, tali caratteri sono elaborati quando si usa l'istruzione POKE per impostare un contatore carattere nella locazione 198. Questi fatti possono essere usati per far si' che una serie di comandi in modo diretto vengano eseguiti automaticamente stampando le istruzioni sullo schermo, inserendo nel buffer un ritorno

carrello, e quindi impostando il contatore carattere. Nell'esempio che segue, il programma lista se' stesso sulla stampante, riattivando successivamente l'esecuzione:

```
10 PRINT CHR$(147)"PRINT#1:CLOSE1:GOTO 50"
20 POKE 631,19:POKE 632,13:POKE 633,13: POKE 198,3
30 OPEN 1,4:CMD1:LIST
40 END
50 REM IL PROGRAMMA RICOMINCIA DA QUI
```

EDITOR DI SCHERMO

L'Editor di Schermo fornisce una serie di agevolazioni potenti e convenienti per l'editazione del testo di un programma. Una volta listato sullo schermo un segmento di programma, si possono usare il tasto cursore ed altri tasti speciali per muovere il cursore sullo schermo per apportare le modifiche desiderate. Dopo aver terminato, per una linea di testo, tutte le correzioni, battendo il tasto RETURN in qualsiasi posizione della linea si ottiene una lettura da parte dell'Editor di Schermo dell'intera linea logica dello schermo (80 caratteri)

Quindi il testo viene passato all'interprete per essere ridotto in simboli ed essere memorizzato nel programma. La linea editata prende in memoria il posto della vecchia versione di quella linea. Si puo' creare una copia addizionale di qualsiasi linea di testo, semplicemente cambiando il numero di linea e premendo RETURN .

Se si usano abbreviazioni delle parole chiave che comportano per quella linea il superamento degli 80 caratteri, al momento di editazione della linea i caratteri in eccesso vengono persi, poiche' l'Editor puo' leggere solo due linee fisiche di schermo. Cio' spiega anche perche' non e' possibile usare l'istruzione INPUT per piu' di 80 caratteri complessivi. Percio', la lunghezza della linea di testo del BASIC, per tutti gli usi pratici, viene limitata a 80 caratteri, come visualizzato sullo schermo.

In particolari condizioni l'Editor di Schermo tratta i tasti di controllo cursore in maniera diversa da quella normale. Se il cursore e' posizionato alla destra di un numero dispari di doppi apici ("), l'Editor opera in MODO VIRGOLETTE.

Questo modo consente ai dati carattere di essere immessi normalmente, ma il controllo cursore, anziche' muovere il cursore, visualizza caratteri "reverse"; lo stesso accade per i tasti di controllo del colore. Pertanto, all'interno di un dato stringa compreso in un programma si possono includere sia i controlli di cursore che i controlli di colore. Infatti, quando viene stampato sullo schermo il testo compreso fra gli apici, il posizionamento del cursore e le funzioni di controllo del colore vengono eseguite automaticamente, facendo parte della stringa. Il controllo cursore puo' essere usato nel modo seguente:

```
Caratteri digitati      10 PRINT"A(R)(R)B(L)(L)(L)C(R)(R)D"
                        20 REM (R)=DESTRA, (L)=SINISTRA

Stampa                  AC BD
```

Il tasto DEL e' il solo controllo cursore non interessato dal modo

virgolette. Se percio' si commette un errore mentre si e' in modo virgolette, non si puo' usare il tasto ⇐CRSR per salvare il resto della frase e riposizionarsi sul punto in cui si e' verificato l'errore - anche il tasto INST produce sul video un carattere reverse. In questo caso, conviene terminare la linea, e dopo aver premuto il tasto RETURN editare la linea normalmente. Come alternativa, se nella stringa non e' richiesto alcun controllo cursore, si possono premere i tasti RUN/STOP e RESTORE che disattivano il modo virgolette. I tasti di controllo cursore utilizzabili in una stringa sono illustrati nella tabella 2.2.

TASTO DI CONTROLLO		VISUALIZZAZIONE
CRSR up	⇑ CRSR	●
CRSR down	CRSR ⇓	Q
CRSR left	⇐CRSR	▌
CRSR right	CRSR⇒	]
CLR	CLR/	♥
HOME	/HOME	S
INST	INST/DEL	T

TABELLA 2.2 – CARATTERI DI CONTROLLO CURSORE NEL MODO VIRGOLETTE

Se NON si e' nel modo virgolette, premendo i tasti SHIFT e INST (INSerT - Inserimento) si verifica uno spostamento dei dati sulla destra del cursore, in modo da aprire uno spazio fra due caratteri per l'inserimento di altri dati. Quindi l'Editor comincia ad operare nel Modo Inserimento, fino al riempimento di tutto lo spazio aperto.

Come nel modo virgolette, anche nel modo inserimento il controllo cursore ed i controlli del colore visualizzano caratteri "reverse". L'unica differenza riguarda il tasto INST/DEL DEL ete INST er T - Cancellazione/Inserimento): DEL, anziche' operare normalmente come nel modo virgolette, visualizza una T reverse, mentre INST, anziche' visualizzare un carattere reverse, inserisce degli spazi come nel modo normale.

Cio' significa che si puo' creare un'istruzione PRINT contenente caratteri DEL, cosa non possibile nel modo virgolette. Il modo inserimento viene disattivato premendo i tasti RETURN SHIFT e RETURN oppure RUN/STOP e RESTORE ; altrimenti, viene disattivato riempiendo tutti gli spazi inseriti. Un esempio di come usare i caratteri DEL in una stringa e' il seguente:

10 PRINT"HELLO" ' DEL INST INST DEL DEL P" (Sequenza di battuta)

10 PRINT"HELP" (Sequenza listata)

Quando viene eseguito questo esempio, viene visualizzata la parola HELP, in quanto le lettere LO sono state cancellate prima di stampare la P. Il carattere DEL all'interno di una stringa funziona sia con LIST che con PRINT: si puo' cosi' "nascondere" una parte o tutta una linea di testo; tuttavia, l'editazione di una linea con questi caratteri e' difficile, se non impossibile.

Per funzioni speciali si possono usare alcuni altri caratteri, anche

se non sono facilmente disponibili da tastiera. Per inserire questi caratteri fra virgolette, occorre lasciare degli spazi vuoti nella linea, premere RETURN ritornare ad editare la linea. Per iniziare a digitare caratteri reverse, premere CTRL e RVS/ON Battere i seguenti tasti:

FUNZIONE DEL TASTO	TASTO PREMUTO	VISUALIZZAZIONE
RETURN shiftato	SHIFT M	
Imposta maiuscolo/minuscolo	N	N
Imposta maiuscolo/grafica	SHIFT N	

Premere i tasti SHIFT e RETURN comporta un ritorno carrello ed un avanzamento linea sullo schermo, ma non termina la stringa. Cio' funziona sia con LIST che con PRINT, per cui l'editazione e' quasi impossibile se si usa questo carattere. Quando si imposta l'output su stampante attraverso l'istruzione CMD, il carattere N "reverse" abilita l'insieme carattere maiuscolo/minuscolo, e , SHIFT N abilita l'insieme carattere maiuscolo/grafica.
I caratteri "reverse" possono essere inclusi in una stringa premendo i tasti CTRL e RVS, provocando la visualizzazione di una R reverse fra virgolette. Cio' fa si' che tutti i caratteri vengano visualizzati in modo reverse (come il negativo di una fotografia). Per terminare la stampa reverse, premere CTRL e RVS/OFF , visualizzando cosi' una R "reverse". I dati numerici possono essere stampati in modo reverse stampando prima CHR$(18); la disabilitazione avviene, in questo caso, stampando CHR$(146) oppure inviando un ritorno carrello.

CAPITOLO 3

programmazione grafica del commodore 64

- Panoramica sulla Grafica
- Locazioni sulla Grafica
- Modo Carattere Standard
- Caratteri Programmabili
- Grafica nel Modo Multicolore
- Modo Colore di Fondo Esteso
- Grafica Bit Map
- Modo Bit Map Multicolore
- Scrolling Rallentato
- Animazioni
- Altre Caratteristiche della Grafica
- Programmazione delle animazioni – Un Ulteriore Sguardo

PANORAMICA SULLA GRAFICA

Tutte le capacita' grafiche del COMMODORE 64 derivano dal Circuito Interfaccia-Video 6567 (conosciuto anche come circuito VIC-II). Questo circuito fornisce una gran quantita' di modi grafici, compresi un video di testo di 40 colonne per 25 righe, un video ad alta risoluzione di 320 per 200 punti, e gli sprite , piccoli oggetti mobili che semplificano la scrittura dei giochi. E se questo non fosse ancora abbastanza, buona parte dei modi grafici possono essere utilizzati contemporaneamente. È possibile, ad esempio, definire la parte alta dello schermo nel modo ad alta risoluzione e quella bassa nel modo testo. E gli sprite si combinano con tutto! Parleremo degli sprite più tardi. Vediamo prima gli altri modi grafici.
Il circuito VIC-II possiede i seguenti modi grafici di visualizzazione:

A) MODO CARATTERE VIDEO

1) Modo Carattere Standard
   a) caratteri Rom
   b) caratteri programmabili della Ram

2) Modo Carattere Multicolore
   a) caratteri Rom
   b) caratteri programmabili della Ram

3) Modo Colore di Fondo esteso
   a) caratteri Rom
   b) caratteri programmabili della Ram

B) MODO BIT MAP

1) Modo Bit Map Standard
2) Modo Bit Map Multicolore

C) ANIMAZIONI

1) Animazioni Standard
2) Animazioni Multicolore

LOCAZIONI DELLA GRAFICA

Innanzi tutto qualche informazione di carattere generale. Sullo schermo del COMMODORE 64 ci sono mille possibili locazioni. Normalmente, lo schermo parte dalla locazione 1024 ($0400 in notazione esadecimale-indicata con HEX) ed arriva alla locazione 2023 è di un Byte. Questo significa che può assumere un valore qualsiasi da 0 a 255. Connesso alla Memoria Video c'è un gruppo di 1000 locazioni chiamate MEMORIA DEL COLORE o RAM DEL COLORE. Queste iniziano alla locazione 55296

(9D800 HEX) e terminano a 56295. Ognuna di queste locazioni occupa 4 bit, cioè $1/2$Byte, può contenere numeri interi da 0 a 15. Di conseguenza ci sono 16 possibili colori a disposizione del COMMODORE 64.
Inoltre, in qualunque istante si possono visualizzare 256 caratteri diversi. Per uno schermo video normale, ognuna delle 1000 locazioni della memoria di schermo contiene un numero di codice che informa il circuito VIC-II su quale carattere visualizzare su quella locazione di schermo.
I vari modi grafici sono selezionati dai 47 registri di CONTROLLO del circuito VIC-II. Gran parte delle funzioni della grafica possono essere controllate introducendo (con l'istruzione POKE) il valore appropriato in uno dei registri. Il circuito VIC-II può essere controllato come detto da 47 registri che hanno come indirizzo di partenza la locazione 53248 ($D000 HEX) e di fine 53294 ($D02E).

SELEZIONE DEL BANCO VIDEO

Il circuito VIC-II è in grado di accedere ("vedere") a soli 16 K di memoria per volta. Poiché ci sono 64 K nella memoria del COMMODORE 64, si puo' desiderare che il circuito VIC-II la scorra per intero. Quello descritto di seguito puo' essere un modo. Ci sono quattro possibili banchi (o segmenti) di 16K di memoria. Tutto cio' che e' necessario e' un mezzo che controlli a quale banco di 16K il circuito VIC-II sta accedendo. I bit di SELEZIONE BANCO, che permettono di accedere a tutti i differenti segmenti di memoria, sono locati nel CIRCUITO #2 ADATTATORE DELL'INTERFACCIA COMPLESSA 6526 (CIA#2). Le istruzioni BASIC PEEK e POKE (o la loro versione in linguaggio macchina) sono usate per scegliere un banco controllando i bit 0 e 1 della PORTA A del CIA#2 [locazione 56576($DD00 HEX)]. Questi due bit devono essere impostati in uscita per cambiare Banchi. Tutto questo e' illustrato nel seguente esempio:

```
POKE56578,PEEK(56578)OR3:REM ASSICURA CHE I BIT 0 E 1 SIANO IMPOSTATI
                          IN  USCITA

POKE 56576,(PEEK(56576)AND252)ORA:REM CAMBIA I BANCHI
```

"A" dovrebbe avere uno dei seguenti valori:

VALORE DI A	BIT	BANCO	LOCAZIONE DI PARTENZA	ESTENSIONE DEL CIRCUITO VIC-II
0	00	3	49152	($C000-$FFFF)*
1	01	2	32768	($8000-$BFFF)
2	10	1	16384	($4000-$7FFF)*
3	11	0	0	($0000-$3FFF) (valore di default)

Questo concetto del Banco di 16 K è solo una parte di tutto ciò che fa il circuito VIC-II. Si dovrebbe sempre conoscere su quale banco il CIRCUITO VIC II, in quanto questo influenza la provenienza delle configurazioni dei dati carattere e delle animazioni, l'origine dello schermo, ecc. All'accensione del COMMODORE 64, i bit 0 e 1 della locazione 56576 sono impostati automaticamente sul BANCO 0 ($0000-$3FFF) per ricevere tutte le informazioni del Video.

* NOTA: L'insieme di caratteri del COMMODRE 64 nei BANCHI 1 e 3 non e'

accessibile al circuito VIC-II (ved. il Capitolo della memoria carattere)

MEMORIA DI SCHERMO

L'indirizzo della memoria di schermo puo' essere modificata semplicemente da una POKE rivolta al registro di controllo 53272 ($D018 HEX). Tuttavia, questo registro e' usato anche per controllare quale insieme di caratteri si e' usato, per cui occorre fare attenzione a non interferire in quella parte del registro di controllo. La locazione della memoria di schermo e' controllata dai bit MAGGIORI di 4. Per rimuovere lo schermo, si deve usare la seguente istruzione:

```
POKE 53272,(PEEK(53272)AND15)ORA
```

dove "A" assume uno dei seguenti valori:

A	BIT	LOCAZIONE (*)	
		DECIMALE	HEX
0	0000XXXX	0	$0000
16	0001XXXX	1024	$0400 (DEFAULT)
32	0010XXXX	2048	$0800
48	0011XXXX	3072	$0C00
64	0100XXXX	4096	$1000
80	0101XXXX	5120	$1400
96	0110XXXX	6144	$1800
112	0111XXXX	7168	$1C00
128	1000XXXX	8192	$2000
144	1001XXXX	9216	$2400
160	1010XXXX	10240	$2800
176	1011XXXX	11264	$2C00
192	1100XXXX	12288	$3000
208	1101XXXX	13312	$3400
224	1110XXXX	14336	$3800
240	1111XXXX	15360	$3C00

* Ricordarsi di sommare l'INDIRIZZO DEL BANCO del Circuito VIC-II

MEMORIA COLORE

La memoria del colore NON puo' essere rimossa. E' sempre contenuta 52296($800 HEX) a 56295($0BE7). La memoria di schermo (1000 locazioni a partire da 1024) e la memoria del colore sono usate in maniera differente nei diversi modi della grafica. Un disegno creato in un modo grafico sembra spesso differente quando viene visualizzato in un altro modo grafico.

MEMORIA CARATTERE

Per la programmazione della grafica e' importante conoscere il punto esatto da cui il VIC-II ottiene le informazioni carattere. Normalmente, il circuito ottiene i caratteri che si vogliono visualizzare dalla ROM GENERATORE DI CARATTERE. In questo circuito sono memorizzate le configurazioni che compongono le varie lettere, numeri, simboli della punteggiatura e tutto quello che e' visibile

sulla tastiera. Una delle caratteristiche del COMMODIORE 64 e' la capacita' di usare le configurazioni locate nella memoria RAM. Queste configurazioni RAM possono essere create dall'utente, per cui si ha un insieme quasi infinito di simboli per giochi, applicazioni commerciali, ecc.

Un normale insieme di caratteri e' composto da 256 caratteri, dove ogni carattere e' definito da 8 bytes di dati. Cio' significa che, occupando ogni carattere 8 bytes, un intero insieme di caratteri occupa 256 X 8 bytes = 2K bytes di memoria. Poiche' il circuito VIC-II osserva ad un istante 16K di memoria, per un insieme di caratteri completo ci sono 8 possibili locazioni. Naturalmente, non si e' obbligati ad usare un intero insieme di caratteri. In ogni caso, questo deve iniziare ad una delle 8 possibili locazioni di partenza.

La locazione della memoria carattere e' controllata da 3 bit del registro di controllo del VIC-II, locati alla posizione 53272($D018 HEX). I bit 3, 2 e 1 controllano in quale blocco di 2K e' locato l'insieme dei caratteri. Il bit 0 e' ignorato. Va ricordato che questo registro e' lo stesso che determina la locazione della memoria di schermo, per cui e' consigliabile evitare di interferire nei bit della memoria di schermo. Per cambiare la locazione della memoria carattere, si puo' usare la seguente istruzione BASIC:

```
POKE53272,(PEEK(53272)AND240)ORA
```

Dove "A" assume uno dei seguenti valori:

VALORE DI A	BIT	LOCAZIONE DELLA MEMORIA CARATTERE (*)	
		DECIMALE	HEX
0	XXXX000X	0	$0000-$07FF
2	XXXX001X	2048	$0800-$0FFF
4	XXXX010X	4096	$1000-$17FF IMMAGINE ROM NEL BANCO 0 & 2 (default)
6	XXXX011X	6144	$1800-$1FFF IMMAGINE ROM NEL BANCO 0 & 2
8	XXXX100X	8192	$2000-$27FF
10	XXXX101X	10240	$2800-$2FFF
12	XXXX110X	12288	$3000-$37FF
14	XXXX111X	14336	$3800-$3FFF

* Ricordarsi di aggiungere l'indirizzo del banco

L'IMMAGINE ROM della tabella precedente si riferisce alla ROM generatore di carattere. Questa compare in sostituzione della RAM nelle precedenti locazioni del banco 0, come pure nella RAM nel banco 2, dalla locazione 36864 ($9000 HEX) alla 40959 ($9FFF HEX). Poiche' il circuito VIC-II, ad un determinato istante, puo' accedere solamente a 16K di memoria, le configurazioni carattere ROM compaiono nel blocco di memoria di 16K osservato dal circuito VIC-II. Percio' il sistema e' stato progettato in modo che il circuito VIC II consideri i caratteri ROM nelle locazioni da 4096 a 8191 ($1000-$1FFF) HEX) per il banco 0, e da 36864 a 40959 ($9000-$9FFF) per il banco 2, anche se la ROM carattere e' attualmente nelle locazioni da 53243 a 57343 ($0000-$DFFF HEX).

Questa rappresentazione e' applicabile solamente al mod di vedere i dati carattere del circuito VIC II. Puo' essere usata per programmi,

altri dati, ecc., in maniera del tutto analoga ad ogni altra memoria RAM.

NOTA: Se tali rappresentazioni ROM fanno parte della grafica impostare ancora i BIT DI SELEZIONE BANCO ad uno dei BANCHI sprovvisti di rappresentazioni ROM (BANCHI 1 O 3).

La locazione ed i contenuti dell'insieme carattere nella ROM sono i seguenti:

BLOCCO	INDIRIZZO		IMMAGINE DEL VIC II	CONTENUTO
	DECIMALE	HEX		
0	53248	D000-D1FF	1000-11FF	Caratteri Maiuscoli
	53760	D200-D3FF	1200-13FF	Caratteri Grafici
	54272	D400-D5FF	1400-15FF	Caratteri Maiuscoli "reverse"
	54784	D600-D7FF	1600-17FF	Caratteri Grafici "reverse"
1	55296	D800-D9FF	1800-19FF	Caratteri Minuscoli
	55808	DA00-DBFF	1A00-1BFF	Caratteri Maiuscoli e grafici
	56320	DC00-DDFF	1C00-1DFF	Caratteri Minuscoli "reverse"
	56832	DE00-DFFF	1E00-1FFF	Caratteri Maiuscoli e Grafici "reverse"

I lettori piu' attenti avranno gia' osservato qualcosa. Le locazioni occupate dalla ROM carattere sono le stesse di quelle occupate dai registri di controllo del circuito VIC-II. Cio' e' possibile in quanto le stesse locazioni non sono occupate nello stesso tempo. Quando il circuito VIC-II ha bisogno di accedere ai dati carattere, la ROM viene posizionata di conseguenza. Essa diviene un'immagine nel banco di memoria di 16K puntato dal circuito VIC-II. Altrimenti, quest'area viene occupata dai registri di controllo dell'I/O, e la ROM carattere rimane a disposizione del circuito VIC-II.
Tuttavia, si puo' aver bisogno di accedere alla ROM carattere nel caso in cui si vogliano usare i caratteri programmabili, e si desideri copiare una parte della ROM carattere per delle definizioni carattere personalizzate. In questo caso si deve escludere il registro di I/O, posizionare la ROM carattere ed eseguire la copia. Al termine, occorre posizionare di nuovo i registri di I/O. Durante la ricopiatura (quando l'I/O e' escluso), non deve avvenire alcuna interruzione, in quanto ai registri di I/O e' demandata la funzione di gestire tale interruzione. Nel caso venga eseguita accidentalmente un'interruzione, la macchina reagira' in maniera anomala. Per escludere la tastiera e le altre normali interruzioni che si verificano sul COMMODORE 64, usare la seguente POKE:

```
POKE56334,PEEK(56334)AND254 (ESCLUDE LE INTERRUZIONI).
```

Al termine del prelievo dei caratteri della ROM carattere, ed al momento dei proseguire con il programma, si deve riattivare la scansione della tastiera con la seguente POKE:

```
POKE56334,PEEK(56334)OR1 (ATTIVA LE INTERRUZIONI)
```

La seguente POKE esclude i I/O e posiziona la ROM CARATTERE:

```
POKE1,PEEK(1)AND251
```

Dopo l'esecuzione di questa istruzione, la ROM carattere si trova nelle locazioni da 53248 a 57343 ($D000-$DFFF).
Per riposizionare l'I/O in $0000 per l'uso in una normale operazione, usare la seguente POKE:

```
POKE1,PEEK(1)OR4
```

MODO CARATTERE STANDARD

Il modo carattere standard e' il modo in cui si trova il COMMODORE 64 al momento dell'accensione per la prima volta. E' il modo in cui di solito si programma.
I caratteri possono essere prelevati dalla ROM o dalla RAM, ma di solito sono prelevati dalla ROM. Quando per un programma si desiderano particolari caratteri grafici, tutto cio' che e' necessario fare e' definire nella RAM le nuove forme carattere, e comunicare al circuito VIC-II di prelevare le informazioni carattere da li' anziche' dalla ROM carattere. Questo argomento e' trattato piu' dettagliatamente nel prossimo paragrafo.
Per visualizzare sullo schermo i caratteri a colori, il circuito VIC-II accede alla memoria schermo allo scopo di determinare il codice carattere per quella locazione sullo schermo. Nello stesso tempo, accede alla memoria del colore per determinare quale colore si desidera per il carattere visualizzato. Il codice carattere viene tradotto dal VIC-II nell'indirizzo di partenza del blocco di 8 byte contenente la configurazione del carattere. Questo blocco di 8 byte e' locato nella memoria carattere. La traduzione non e' molto complicata, anche se per generare l'indirizzo desiderato vengono combinate diverse voci. Innanzitutto, il codice carattere usato per modificare (POKE) la memoria schermo viene moltiplicato per 8. Al risultato viene aggiunto l'indirizzo di partenza della memoria carattere (vd. il paragrafo MEMORIA CARATTERE). Successivamente vengono presi in considerazione i bit di SELEZIONE BANCO, che vengono sommati all'indirizzo base. (vd. il paragrafo SELEZIONE DEL BANCO VIDEO). La formula seguente illustra quanto sopra esposto:

INDIRIZZO CARATTERE=CODICE SCHERMO * 8 + (INSIEME CARATTERE*2048) + (BANCO * 16384)

DEFINIZIONE DEL CARATTERE

Ogni carattere e' formato da una griglia (matrice) di 8 punti X 8, ogni punto della quale puo' essere acceso (ON) o spento (OFF). Le immagini carattere del COMMODORE 64 sono registrate nel circuito ROM GENERATORE DI CARATTERI. I caratteri sono registrati come insiemi di 8 byte per ogni carattere, dove ogni di questi bytes bit rappresenta. Un punto. Un bit a 0 indica che il punto è spento, e un bit a 1 indica che il punto è 0 acceso. Nella Rom, la memoria carattere inizia alla locazione 53248 (quanto l' I/O e' escluso). I primi 8 byte, dalla locazione 53248 ($D000 HEX) alla 53255 ($D007 HEX) contengono la configurazione del simbolo @, il cui codice carattere ha valore 0 nella memoria di schermo Gli 8 bytes successivi, dalla locazione 53256 ($D008) alla 53263 ($D00F), contengono le informazione necessarie alla costruzione della lettera "A".

IMMAGINE	BINARIO	PEEK
**	00011000	24
****	00111100	60
** **	01100110	102
******	01111110	126
** **	01100110	102
** **	01100110	102
** **	01100110	102
	00000000	0

Ogni insieme carattere completo occupa 2k di memoria (2048 bit), pari a 256 caratteri di 8 byte ciascuno. Poiche' ci sono due insiemi carattere, uno per'le lettere maiuscole e la grafica, l'altro per le lettere maiuscole e minuscole, la ROM generatore di caratteri occupa in totale 4K locazioni.

CARATTERI PROGRAMMABILI

Poiche' i caratteri sono memorizzati nella ROM, sembrerebbe a prima vista non esserci modo di cambiarli a favore di caratteri definiti dall'Utente. Tuttavia, la locazione di memoria che indica al circuito VIC-II dove trovare i caratteri e' un registro programmabile che puo' essere modificato per puntare ad una parte qualsiasi della memoria. Modificando il puntatore alla memoria carattere in modo che esso punti alla RAM, l'insieme carattere può essere programmato per ogni necessità.

Se si vuole locare in RAM un'insieme carattere, ci sono pochi punti MOLTO IMPORTANTI da prendere in considerazione all'atto della programmazione di un insieme carattere personalizzato. Inoltre, ci sono altri due punti importanti di cui si deve essere a conoscenza per creare caratteri speciali personalizzati:

1) E' un operazione "tutto o niente". Di solito, se si usa un insieme carattere personalizzato per comunicare al circuito VIC-II che l'informazione carattere e' disponibile nell'apposita area RAM, i 64 caratteri standard Commodore non sono accessibili. Questo e' risolvibile copiando nell'apposita memoria carattere RAM tutte le lettere, i numeri o la grafica standard del COMMODORE 64 che si intende usare. Si possono scegliere e prendere solo i caratteri che si desiderano, e senza neanche metterli in ordine!

2) Il nuovo set di caratteri programmato dall'utente toglie spazio di memoria al programma BASIC. Naturalmente, con 38K disponibili per un programma BASIC, quasi tutte le applicazioni non danno problemi.

ATTENZIONE: Occorre fare attenzione nel proteggere il set di caratteri dalla sovrapposizione del programma BASIC, visto che anch'esso usa la RAM.

Nel COMMODORE 64 ci sono due locazioni da cui far partire l'insieme carattere personalizzato che NON DEVONO ESSERE USATE CON IL BASIC: locazione 0 e locazione 2048. La prima non deve essere usata in quanto il sistema memorizza dati importanti sulla pagina 0 la seconda non puo' essere usata poiche' da tale locazione ha origine il programma

BASIC! Ci sono pero' sei altre posizioni per il set di caratteri personalizzato. Il punto di partenza migliore in cui porre il set di caratteri personalizzato per l'uso con il BASIC durante le prove e' la locazione 12288($3000 HEX). Questo viene fatto modificando (tramite l'istruzione POKE) i quattro bit piu' bassi della locazione 53272 con 12. Un esempio puo' essere:

```
POKE53272,(PEEK(53272)AND240)+12
```

Subito, tutte le lettere sullo schermo si trasformano in caratteri indecifrabili in quanto finora non c'è alcun insieme di caratteri alla locazione 12288... solo bytes in ordine casuale. Eseguire il ripristino delle compizioni iniziali del COMMODORE 64 al normale premendo il tasto RUN/STOP e quindi RESTORE

Iniziamo ora la creazione della grafica. Per proteggere l'insieme carattere dal BASIC e' opportuno ridurre la quantita' di memoria che il BASIC ritiene di avere a disposizione. La quantita' di memoria del Computer rimane la stessa......Soltanto che al BASIC viene impedito l'utilizzo di una parte di essa. Ad esempio:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Il numero visualizzato e' la quantita' di memoria inutilizzata. Battere ora:

```
POKE52,48:POKE56,48:CLR
```

e quindi:

```
PRINT FRE(0)-(SGN(FRE(0))<0)*65535
```

Che cosa è cambiato? I risultati delle operazioni di FRE sono diverse, ora il BASIC ha a disposizione meno memoria. La memoria appena tolta al BASIC è quella dove va a risiedere il set caratteri, al riparo dalle azioni del BASIC. Il passo successivo è introdurre i caratteri nella Ram. All'inizio, partendo dalla locazione 12288 ($3000 HEX) si devono introdurre nelle RAM le configurazioni carattere (analoghe a quelle residenti in ROM) a disposizione del circuito VIC-II. Il programma seguente trasporta 64 caratteri dalla ROM alla RAM riservata all'insieme carattere.

```
5 PRINTCHR$(142) : REM IMPOSTA LE MAIUSCOLE
10 POKE52,48:POKE56,48:CLR : REM RISERVA MEMORIA PER CARATTERI
20 POKE56334,PEEK(56334 )AND254 : REM DISATTIVA IL TIMER DI
21 REM INTERRUZIONE DELLA TASTIERA
30 POKE1,PEEK(1)AND251 : REM ATTIVA UN CARATTERE
40 FORI=0TO511:POKEI+12288,PEEK(I+53248 ):NEXT
50 POKE(1),PEEK(1)OR4 : REM ATTIVA L'I/O
60 POKE56334,PEEK(56334 )OR1 : REM RIATTIVA IL TIMER DI
61 REM INTERRUZIONE DELLA TASTIERA
70 END
```

Modifichiamo ora la locazione 53272 con (PEEK(53272)AND240)+12. Che cosa succede? Beh, quasi nulla. Ora il COMMODORE 64 sta traendo l'informazione carattere dalla RAM anziche' dalla ROM. Ma finche' i

caratteri non sono stati copiati esattamente dalla ROM, non si puo' osservare alcuna differenza.
A questo punto si possono finalmente modificare i caratteri. Azzerare lo schermo e battere @. Abbassare il cursore di alcune linee, e battere:

```
FORI=12288TO12288+7:POKE1,255-PEEK(I).NEXT
```

Si e' appena creato una @ in campo "reverse"!

INFORMAZIONE: I caratteri in campo inverso sono solamente caratteri le cui configurazioni di bit nella memoria carattere sono state invertite.

Spostare ora il cursore di nuovo sul programmma e premere di nuovo RETURN per invertire un'altra volta il carattere (riportandolo a normale). Osservando la tavola dei codici dello schermo video, ci si puo' rendere conto di quale posizione occupi ogni carattere nella RAM. Da notare solamente che ogni carattere sottrae alla memoria 8 locazioni. Qui di seguito sono dati alcuni esempi:

CARATTERI	CODICE VIDEO	LOCAZIONE D'INIZIO DELLA RAM
@	0	12288
A	1	12296
?	33	12552
>	62	12784

Va ricordato che abbiamo usato solo i primi 64 caratteri. Se si desiderano altri caratteri, occorre fare qualcos'altro. Ad esempio, cosa andrebbe fatto se si desiderasse il carattere 154, una Z "reverse"?
Si potrebbe semplicemente battere una Z "reverse", oppure copiare dalla ROM l'insieme dei caratteri in campo inverso, o anche prelevare dalla ROM il carattere desiderato e sostituirlo nella RAM ad uno dei caratteri che non vengono usati.
Supponiamo di non usare il segno ">", e sostituiamolo con Z "reverse". Digitare:

```
FORI=0TO7:POKE12784+I,255-PEEK(I+12496):NEXT
```

Digitare ora il segno >: comparira' una Z "reverse". Tutte le volte che verra' battuto >, comparira' una Z "reverse" (in realta' questo scambio non avviene. Anche se il segno > fa comparire una Z in campo inverso, in un programma agisce ancora come >. Trovando qualcosa che richieda il segno >, si osservera' che funziona ancora bene, solo che sembrera' strano).

RAPIDO RIASSUNTO: a questo punto si e' in grado di copiare caratteri dalla ROM alla RAM, perfino di sceglere quelli che vogliamo. Rimane soltanto un passo da compiere nei caratteri programmabili (cioe' il migliore!).....costruire caratteri personalizzati. Ogni carattere e' memorizzato nella ROM come un gruppo di 8 byte. Le configurazioni di bit dei byte controllano direttamente il carattere. Se si incolonnano 8 byte, e si scrive ogni byte come una seguenza di 8 numeri binari, si costruisce una matrice 8 X 8, che somiglia ai caratteri. Quando un bit

e' a 1, in quella locazione c'e' un punto, quando un bit e' a 0 in quella locazione c'e' uno spazio. Quando si creano caratteri personalizzati, in memoria viene impostato lo stesso tipo di tabella. digitare NEW e poi il seguente programma:

```
10FORI=12448TO12455:READA:POKEI,A:NEXT
20DATA60,66,165,129,165,153,66,60
```

Ora battere RUN. Il programma sostituisce la lettera T con una faccia sorridente. Ogni numero dell'istruzione DATA della linea 20 corrisponde ad una riga dalla faccia sorridente La matrice della faccia e' simile a questa:

		7	6	5	4	3	2	1	0	BINARIO	DECIMALE
RIGA	0			.	.	.	.			00111100	60
	1		.					.		01000010	66
	2	.		.			.		.	10100101	165
	3	.							.	10000001	129
	4	.		.			.		.	10100101	165
	5	.			.	.			.	10011001	153
	6		.					.		01000001	66
RIGA	7			.	.	.	.			00111100	60

	7	6	5	4	3	2	1	0
0								
1								
2								
3								
4								
5								
6								
7								

FIGURA 3.1. Tabella per la costruzione di caratteri programmabili

La tabella per la costruzione di caratteri programmabili (figura 3.1) puo' aiutare a disegnare i caratteri. Sul foglio c'e' una matrice 8 X 8, con i numeri di riga e di colonna (se si considera ogni riga come una parola binaria, i numeri sono il valore di quella posizione del bit. Ogni numero e' una potenza di due. Il bit piu' a sinistra e' uguale a 128 (=2 elevato alla potenza 7), il successivo e' uguale a 64 (=2 elevato alla potenza 6) e cosi' via, finche' non si raggiunge il bit piu' a destra che e' uguale ad 1 (=2 elevato alla potenza 0).
Tracciare sulla matrice una X in ogni locazione dove si vuole un punto nel carattere. Quando il carattere e' pronto si puo' creare l'istruzione DATA per quel carattere.
A partire dalla prima riga, scrivere il numero in cima ad ogni

colonna dove sia stata piazzata una X (tale numero, come spiegato sopra, e' una potenza di 2). Completata la prima riga, sommare tali numeri e scrivere il risultato vicino alla riga. Questo e' il numero da inserire nell'istruzione DATA per disegnare tale riga.

Operando allo stesso modo con le altre 7 righe, si ottengono 8 numeri compresi fra 0 e 255. Per essere corretti, tali numeri devono essere compresi in questo intervallo! Se si hanno meno di 8 numeri si e' saltata una riga. Se alcuni sono 0 va bene. La riga 0 ha la stessa importanza delle altre. Sostituire i numeri dell'istruzione DATA alla linea 20 con quelli appena calcolati, e far girare (RUN) il programma. Poi battere una T. Ogni volta che questo tasto viene battuto, compare il carattere personalizzato.

Se il carattere risultante non e' riuscito bene, basta cambiare i numeri nell'istruzione DATA e rilanciare (RUN) il programma finche' non si e' soddisfatti. Questo e' tutto!

SUGGERIMENTO: per ottenere migliori risultati si consiglia di fare le linee verticali ampie almeno due punti (bit). Cio' previene il disturbo CHROMA (distorsione del colore) quando i caratteri vengono visualizzati su uno schermo TV.

Il seguente esempio illustra un programma che usa i caratteri programmabili standard:

```
10 REM * ESEMPIO 1 *
20 CREAZIONE DI CARATTERI PROGRAMMABILI
30 REM DISATTIVA KB E I/O
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM INTERVALLO CARATTERI DA COPIARE DALLA ROM
36 FORJ=0TO7:REM COPIA TUTTI GLI 8 BYTE DI UN CARATTERE
37 POKE12288+I*8+J,PEEK(12288+I*8+J):REM COPIA UN BYTE
38 NEXTJ:NEXTI:REM PASSA AL PROSSIMO BYTE O CARATTERE
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM ATTIVA I/O E KB
40 POKE53272,(PEEK(53272)AND240)+12:REM IMPOSTA IL PUNTATORE
41 REM CARATTERE ALLA LOCAZIONE DI MEMORIA 12288
60 FORCHAR=60TO63:REM PROGRAMMA I CARATTERI DA 60 A 63
80 FORBYTE=0TO7:REM COSTRUISCE TUTTI GLI 8 BYTE DI UN CARATTERE
100 READ NUMBER:REM LEGGE 1/8 DEI DATI DI UN CARATTERE
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM MEMORIZZA I DATI
140 NEXTBYTE:NEXTCHAR:REM PASSA AL PROSSIMO BYTE O CARATTERE
150 PRINTCHR$(147)TAB(255)CHR$(60);
155 PRINTCHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 150 VISUALIZZA I CARATTERI APPENA DEFINITI
170 GETA$:REM ATTENDE CHE L'UTENTE PREMA UN TASTO
180 IFA$=""THENGOTO170:REM SE NON SI PREMONO TASTI, ATTENDE DI NUOVO!
190 POKE53272,21:REM RITORNA AI CARATTERI NORMALI
200 DATA4,6,7,5,7,7,3,3:REM DATI DEL CARATTERE 60
210 DATA32,96,224,160,224,224,192,192:REM DATI DEL CARATTERE 61
220 DATA7,7,7,31,31,95,143,127:REM DATI DEL CARATTERE 62
230 DATA224,224,224,248,248.248,240,224:REM DATI DEL CARATTERE 63
240 END
```

GRAFICA DEL MODO MULTICOLORE

La grafica standard ad alta risoluzione da' il controllo su punti molto piccoli dello schermo. Ogni punto della memoria carattere puo' assumere due valori, 1 per acceso e 0 per spento. Quando un punto e' spento, il colore dello schermo viene usato sullo spazio riservato a quel punto. Se il punto e' acceso, viene colorato con il colore del carattere che si e' scelto per quella posizione dello schermo. Quando si usa la grafica standard ad alta risoluzione,tutti i punti interni ad un carattere 8 X 8 possono avere sia il colore di fondo che il colore principale. Questo limita in qualche modo la risoluzione del colore all'interno dello spazio. Ad esempio, possono insorgere dei problemi quando s'incrociano due linee di differenti colori.

Il modo multicolore risolve questo problema. Ogni punto nel MODO MULTICOLORE puo' essere di quattro colori: colore di schermo (registro #0 del colore di fondo), colore contenuto nel registro #1 di fondo, colore contenuto nel registro #2 di fondo, o colore carattere. L'unico sacrificio e' a carico della risoluzione orizzontale, in quanto ogni punto del modo multicolore e' largo il doppio di un punto ad alta risoluzione. Questa minima perdita di risoluzione e' largamente compensata dalle alte capacita' del modo multicolore.

BIT DEL MODO MULTICOLORE

Per attivare il Modo Carattere Multicolore, impostare a 1 il BIT 4 del registro di controllo del VIC-II situato nella locazione 53270 ($D016) usando la seguente POKE:

```
POKE 53270,PEEK(53270)OR16
```

Per disattivare il modo carattere multicolore, impostare a 0 il bit 4 della locazione 53270 per mezzo della seguente POKE:

```
POKE53270,PEEK(53270)AND239
```

Il modo multicolore e' impostato a ON o OFF oer ogni spazio dello schermo, in modo che la grafica multicolore possa essere usata insieme alla grafica ad alta risoluzione (hi-res). Cio' viene controllato dal BIT 3 della memoria del colore, che inizia alla locazione 55296 ($D800 HEX). Se il numero della memoria del colore e' minore di 8 (0-7), lo spazio corrispondente sullo schermo video viene considerato standard ad alta risoluzione, nel colore (0-7) scelto. Se il numero locato nella memoria del colore e' maggiore o uguale a 8 (8-15), allora lo spazio viene visualizzato nel modo multicolore. Introducendo con una POKE un numero nella memoria del colore, si puo' cambiare il colore del carattere in una data posizione dello schermo. Un numero compreso fra 0 e 7 da' colori carattere normale. Un numero compreso fra 8 e 15 converte lo spazio nel modo multicolore. In altri termini, mettendo a ON il bit 3 della memoria colore, si imposta il MODO MULTICOLORE, mentre mettendo lo stesso bit a OFF viene impostato il modo normale ad ALTA RISOLUZIONE. Una volta che uno spazio e' impostato al modo multicolore, i bit del carattere determinano con quali colori devono essere visualizzati i punti. L'esempio seguente illustra la

costruzione della lettera A e la sua configurazione di bit:

```
 IMMAGINE        CONFIGURAZIONE BIT
    * *              00011000
   *****             00111100
  **   **            01100110
  *******            01111110
  **   **            01100110
  **   **            01100110
  **   **            01100110
                     00000000
```

Nel modo normale o ad alta risoluzione,il colore dello schermo e' visualizzato per tutti i bit impostati a 0, mentre il colore carattere viene visualizzato per tutti i bit impostati a 1. Il modo multicolore usa coppie di bit, come nell'esempio seguente:

```
 IMMAGINE        CONFIGURAZIONE BIT
   AABB             00 01 10 00
   CCCC             00 11 11 00
 AABBAABB           01 10 01 10
 AACCCCBB           01 11 11 10
 AABBAABB           01 10 01 10
 AABBAABB           01 10 01 10
 AABBAABB           01 10 01 10
                    00 00 00 00
```

Nella precedente area immagine, gli spazi indicati con AA sono tracciati con il colore di fondo #1, quelli indicati con BB usano il colore di fondo #2, e quelli indicati con CC usano il colore carattere. Tutto cio' viene determinato dalle coppie di bit, secondo quanto illustrato nella seguente tabella:

COPPIA DI BIT	REGISTRO COLORE	LOCAZIONE
00	Colore #0 di fondo (colore schermo)	53281 ($D021)
01	Colore #1 di fondo	53282 ($D022)
10	Colore #2 di fondo	53283 ($D023)
11	Colore specificato dai 3 bit bassi della memoria del colore	RAM colore

Battere NEW e poi il seguente programma dimostrativo:

```
100 POKE53281,1:REM IMPOSTA A BIANCO ILCOLORE #0 DI FONDO
110 POKE53282,3:REM IMPOSTA AD AZZURRO IL COLORE #1 DI FONDO
120 POKE53283,8:REM IMPOSTA AD ARANCIO IL COLORE #2 DI FONDO
130 POKE53270,PEEK(53270)OR16:REM ATTIVA IL MODO MULTICOLORE
140 C=13*4096+8*256:REM IMPOSTA C PER PUNTARE ALLA MEMORIA COLORE
150 PRINTCHR$(147)"AAAAAAAAAA"
160 FORL=0TO9
170 POKEC+L,8
180 NEXT
```

Il colore di schermo e' bianco, il colore carattere e' nero, un registro colore e' azzurro, l'altro e' arancione.
In realta', nello spazio considerato non vengono introdotti i codici di colore al posto del colore carattere, piuttosto si fanno dei riferimenti ai registri associati a quei colori. Si ha cosi' un risparmio di memoria, in quanto si usano due soli bit per scegliere 16 colori di fondo oppure 8 colori carattere. Si possono inoltre realizzare alcuni trucchetti: semplicemente modificando uno dei registri indiretti si puo' cambiare ogni punto tracciato con quel colore. Percio', tutto quanto e' stato tracciato con i colori di schermo e di fondo puo' essere istantaneamente modificato sull'intero schermo. Quello seguente e' un esempio di come viene modificato il registro colore di fondo #1:

```
100 POKE53270,PEEK(53270)OR16:REM ATTIVA IL MODO MULTICOLORE
110 PRINTCHR$(147)CHR$(18);
120 PRINT"" ;:REM BATTE C= & 1 PER ARANCIO O PER FONDO MULTICOLORE
125 REM           NERO
130 FOR=1TO22:PRINTCHR$(65);:NEXT
135 FORT=1TO500:NEXT
140 PRINT"" ;:REM BATTE CTRL & 7 PER CAMBIARE IL COLORE IN BLU
145 FORT=1TO500:NEXT
150 PRINT" BATTI UN TASTO"
160 GETA$:IFA$=""THEN160
170 X=INT(RND(1)*16)
180 POKE53282,%
190 GOTO160
```

Usando il tasto C= ed i tasti colore, i caratteri assumono qualunque colore, compresi i caratteri multicolore. Battere ad esempio il seguente comando:

```
POKE53270,PEEK(53270)OR16:PRINT"";(rosso acceso/rosso multicolore)
```

La parola READY e qualunque altra parola battuta viene visualizzata nel modo multicolore. Un altro controllo di colore riporta al testo regolare. Il seguente programma illustra l'uso dei caratteri multicolore programmabili:

```
10 REM *ESEMPIO 2*
20 REM CREAZIONE DI CARATTERI PROGRAMMABILI MULTICOLORE
31 POKE56334,PEEK(56334)AND254:POKE1,PEEK(1)AND251
35 FORI=0TO63:REM INTERVALLO CARATTERI DA COPIARE DALLA ROM
36 FORJ=0TO7:REM COPIA TUTTI GLI 8 BYTE DI UN CARATTERE
37 POKE12288+I*8+J,PEEK(53248+I*8+J):REM COPIA UN BYTE
38 NEXTJ,I:REM PASSA AL PROSSIMO BYTE O AL PROSSIMO CARATTERE
39 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1:REM ATTIVA I/O E KB
40 POKE53272,(PEEK(53272)AND240)+12:REM IMPOSTA PUNTATORE CARATTERE
45 REM                                  ALLA LOCAZIONE 12288
50 POKE53270,(PEEK(53270)OR16
51 POKE53281,0:REM IMPOSTA A NERO IL COLORE DI FONDO #0
52 POKE53282,2:REM IMPOSTA A ROSSO IL COLORE DI FONDO #1
53 POKE53283,7:REM IMPOSTA A GIALLO IL COLORE DI FONDO #2
60 FORCHAR=60TO63:REM PROGRAMMA I CARATTERI DA 60 A 63
```

```
80 FORBYTE=0TO7:REM PREPARA TUTTI GLI 8 BYTE DI UN CARATTERE
100 READNUMBER:REM LEGGE 1/8 DEI DATI DI UN CARATTERE
120 POKE12288+(8*CHAR)+BYTE,NUMBER:REM MEMORIZZA I DATI
140 NEXTBYTE,CHAR
       SHIFT CLR/HOME
150 PRINT"♥"TAB(255)CHR$(60)CHR$(61)TAB(55)CHR$(62)CHR$(63)
160 REM LA LINEA 50 VISUALIZZA I NUOVI CARATTERI APPENA DEFINITI
170 GETA$:REM ATTENDE LA PRESSIONE DI UN TASTO
180 IFA$=""THEN170:REM SE NESSUN TASTO VIENE BATTUTO, RICOMINCIA
190 POKE53272,21:POKE53270,PEEK(53270)AND239
191 REM RITORNA AI CARATTERI NORMALI
200 DATA129,37,21,29,93,85,85,85:REM DATI CARATTERE 60
210 DATA66,72,84,116,117,85,85,85:REM DATI CARATTERE 61
220 DATA87,87,85,21,8,8,40,0:REM DATI CARATTERE 62
230 DATA213,213,85,84,32,32,40,0:REM DATI CARATTERE 63
240 END
```

MODO COLORE DI FONDO ESTESO

Il modo colore di fondo esteso controlla oltre al colore principale, il colore di fondo di ogni singolo carattere. In questo modo, ad esempio, si riesce a visualizzare su uno schermo bianco un carattere blu su fondo giallo.

Il modo colore di fondo esteso ha a disposizione 4 registri, ognuno dei quali puo' essere impostato con uno dei 16 colori.

La memoria del colore viene usata per contenere il colore di fondo nel modo di fondo esteso. Viene usata analogamente a quanto visto per il modo carattere standard.

Il modo carattere esteso, tuttavia, pone un limite al numero di differenti caratteri visualizzabili. Quando si e' nel modo colore esteso, si possono usare solamente i primi 64 caratteri della ROM carattere (oppure i primi 64 caratteri dell'insieme caratteri programmabili). Questo perche' per la selezione del colore di fondo si usano 2 bit del codice carattere. Questo concetto puo' essere espresso anche come segue:

Il codice carattere (il numero da inviare allo schermo con una POKE) della lettera "A" e' 1. Quando si e' nel modo colore esteso, se si invia allo schermo un 1, appare una "A". Se si inviasse normalmente allo schermo un 65, ci si potrebbe aspettare la comparsa di un carattere il cui codice carattere e' 129, cioe' una A "reverse". Nel modo colore esteso questo NON avviene: compare infatti la stessa A "reverse" di prima, ma visualizzata su un diverso colore di fondo. Tutti i codici sono riassunti nella seguente tabella:

CODICE CARATTERE			REGISTRO DEL COLORE DI FONDO	
INTERVALLO	BIT 6	BIT 7	NUMERO	INDIRIZZO
0-63	0	0	0	53281($D021)
64-127	0	1	1	53282($D022)
128-191	1	0	2	53283($D023)
192-255	1	1	3	53284($D024)

Il modo colore esteso viene attivato (ON) impostando a 1 il BIT 6 del registro del VIC-II contenuto nella locazione 53265 ($D011 HEX), tramite la seguente POKE:

```
POKE53265,PEEK(53265)OR64
```

GRAFICA «BIT MAP»

Quando si scrivono giochi, grafici commerciali o altri tipi di programmi, presto o tardi ci si imbatte nella necessita' di una visualizzazione ad alta risoluzione.
Il COMMODORE 64 e' stato progettato proprio per questo: l'alta risoluzione e' disponibile per mezzo della scansione bit map dello schermo. Con questo metodo e' possibile assegnare ad ogni possibile punto di risoluzione (pixel) dello schermo un bit (locazione) in memoria. Se questo bit di memoria e' a 1 , il punto a cui viene assegnato e' acceso, se invece il bit e' a 0 il punto e' spento.
Il modello della grafica ad alta risoluzione presenta un paio di inconvenienti, che spiegano perche' non viene usata appieno. Innanzi tutto, scandire punto a punto l'intero schermo richiede una notevole quantita' di memoria in quanto un simile controllo richiede un bit di memoria per ogni pixel. Poiche' ogni carattere e' una matrice di 8 bit X 8 e ci sono 25 linee di 40 caratteri l'una, la risoluzione per l'intero video e' data da 320 pixel (punti) per 200 pixel, per un totale di 64000 punti distinti, ognuno dei quali richiede un bit di memoria. In altri termini, per tracciare l'intero video sono necessari 8000 byte.
Generalmente, le operazioni ad alta risoluzione sono composte da molte procedure brevi, semplici e ripetitive che purtroppo rallentano la stesura di un programma in BASIC. Tuttavia, questo tipo di procedure viene gestito al meglio dal linguaggio macchina. La soluzione e' quindi scrivere i programmmi interamente in linguaggio macchina, oppure richiamare il linguaggio macchina, usando da BASIC il comando SYS per le sottoprocedure ad alta risoluzione. Si ottine cosi', per la grafica, sia la semplicita' della stesura in BASIC, sia la velocita' del linguaggio macchina. Per raggiungere i comandi ad alta risoluzione al BASIC del COMMODORE 64 e' disponibile anche la CARTUCCIA VSP.
Tutti gli esempi di questo paragrafo vengono dati in Basic per chiarire questi concetti. Passiamo ora ai dettagli tecnici.
IL BIT MAPPING e' una delle tecniche di grafica piu' diffuse; viene usata per creare figure molto dettagliate. Fondamentalmente, quando il COMMODORE 64 entra nel modo bit map, automaticamente visualizza sullo schermo TV una sezione di memoria di 8 K; nel modo bit map si puo' controllare direttamente se un singolo punto del video e' acceso o spento. A disposizione del COMMODORE 64 ci sono due tipi di "bit mapping":

1)MODO BIT-MAP STANDARD(alta risoluzione da 320 X 200 punti)

2)MODO BIT-MAP MULTICOLORE(risoluzione da 160 X 200 punti).

Entrambi sono molto simili al tipo carattere con cui sono stati chiamati: il modo bit map standard ha maggiore risoluzione ma meno possibilita' di colore, il modo bit map multicolore compensa la risoluzione orizzontale con una grande possibilita' di colori per un quadrato di 8 punti X 8.

MODO BIT MAP STANDARD AD ALTA RISOLUZIONE

Il modo bit map standard ha una risoluzione di 320 punti orizzontali per 200 verticali, per una scelta di 2 colori in ogni segmento di 8 X 8 punti. Il modo bit map viene selezionato (ON) impostando a 1 il bit 5 del registro di controllo del VIC-II locato alla posizione 53265 ($D011 HEX), tramite la seguente POKE:

```
POKE 53265,PEEK(53265)OR32
```

Analogamente, il modo bit map viene disattivato (OFF) impostando a 0 il bit 5 dello stesso registro tramite la seguente POKE:

```
POKE53265,PEEK(53265)AND223
```

Prima di entrare nel dettaglio del modo bit map c'e' ancora un punto da apprendere: dove locare l'area per il bit map.

FUNZIONAMENTO

Nel paragrafo riguardante i caratteri programmabili si e' detto che e' possibile impostare a piacere la configurazione di bit di un carattere memorizzato nella RAM. Se dunque si riesce a cambiare il carattere visualizzato sullo schermo, si e' anche in grado di modificare un singolo punto. Questo e'il concetto fondamentale del "bit mapping". L'intero schermo puo'essere riempito di caratteri programmabili,e le modifiche avvengono direttamente nella memoria da cui i caratteri programmabili traggono le configurazioni.
Ciascuna locazione della memoria di schermo, precedentememte usata per controllare il carattere visualizzato, viene ora utilizzata per l'informazione del colore. La locazione 1, ad esempio, anziche' contenere 1 per far apparire una "A" nell'angolo in alto a sinistra dello schermo, controlla il colore del bit di quello stesso spazio.
Al contrario di quanto avviene nei modi carattere, nel modo bit map i colori dei quadrati non provengono dalla memoria del colore, bensi'dalla memoria dello schermo. I 4 bit piu' alti della memoria di schermo diventano il colore di tutti i bit impostati a 1 nell'area 8x8 controllata da quella locazione della memoria di schermo. I 4 bit piu' bassi diventano il colore di tutti i Bit impostati a 0.

ESEMPIO – Battere:

```
5 BASE =2*4096:POKE53272,PEEK(53272)OR8:REM PONE IL BIT MAP A 8192
10 POKE53265,PEEK(53265)OR:32:REM ENTRA NEL MODO BIT MAP
```

Lanciare (RUN) il programma: sullo schermo compaiono caratteri indecifrabili. Come nel modo schermo normale, si e' azzerato lo schermo ad ALTA RISOLUZIONE (HI-RES) prima di usarlo. In questo caso, pero',non serve a niente battere CLR: si deve infatti azzerare la sezione di memoria usata per i caratteri programmabili. Battere i tasti RUN/STOP e RESTORE , e poi aggiungere alle precedenti le seguenti linee per azzerare lo schermo HI-RES:

```
20 FORI=BASETOBASE+7999:POKEI,0:NEXT:REM AZZERA IL BIT MAP
30 FORI=1024TO2023:POKEI,3:NEXT:REM IMPOSTA I COLORI AZZURRO E NERO
```

Una volta rilanciato il programma, si puo' osservare l'azzeramento

dello schermo, e successivamente l'azzurro che ricopre l'intero schermo. Vediamo a questo punto come attivare e disattivare punti sul video HI-RES.

Per attivare (ON) e disattivare (OFF) un punto occorre conoscere come trovare nella memoria carattere il bit impostato A 1. In altre parole, si deve trovare il carattere, quale riga di quel carattere e quale bit di quella riga devono essere cambiati. A tale scopo si puo'usare una formula.

Indichiamo con X e Y rispettivamemte la posizione orizzontale e verticale di un punto: il punto dove X=0 e Y=0 si trova nell'angolo in alto a sinistra del video. I valori di X aumentano da sinistra verso destra, mentre i valori di Y aumentano dall'alto verso il basso.

Il modo migliore di usare il bit map e'considerare il video strutturato come nella seguente figura:

```
0----------------------X--------------------------------319
.
.
.
Y
.
.
.
199----------------------------------------------------------
```

Anche se la rappresentazione attuale e'la seguente:

```
------BYTE 0   BYTE 8   BYTE 16  BYTE 24  ..........BYTE 312
L  R  BYTE 1   BYTE 9     ..       ..               BYTE 313
I  I  BYTE 2   BYTE 10    ..       ..               BYTE 314
N  G  BYTE 3   BYTE 11    ..       ..               BYTE 315
E  A  BYTE 4   BYTE 12    ..       ..               BYTE 316
A  0  BYTE 5   BYTE 13    ..       ..               BYTE 317
1     BYTE 6   BYTE 14    ..       ..               BYTE 318
------BYTE 7   BYTE 15    ..       ..               BYTE 319

------BYTE 320 BYTE 328 BYTE 336 BYTE 344 .......  BYTE 632
L  R  BYTE 321 BYTE 329 ..         ..               BYTE 633
I  I  BYTE 322 BYTE 330 ..         ..               BYTE 634
N  G  BYTE 323 BYTE 331 ..         ..               BYTE 635
E  A  BYTE 324 BYTE 332 ..         ..               BYTE 636
A  1  BYTE 325 BYTE 333 ..         ..               BYTE 637
2     BYTE 326 BYTE 334 ..         ..               BYTE 638
------BYTE 327 BYTE 335 ..         ..               BYTE 639
```

I caratteri programmabili di cui e' composta la bit map sono disposti in 25 righe di 40 colonne ciascuna: questa rappresentazione, che e' un buon metodo di organizzazione per il testo, presenta invece alcune difficolta' per quanto concerne il bit map (vedere il paragrafo Modi

Misti).
La seguente formula semplifica il controllo di un punto sullo schermo del bit map:
L'inizio dell'area di memoria del video si chiama BASE. La riga (da 0 24) su cui si trova un carattere e' data da:

ROW = INT(Y/8) (320 BYTE per riga)

La colonna (da 0 a 39) su cui si trova un carattere e' data da:

CHAR = INT(Y/8) (8 BYTE per colonna)

La linea della posizione (da 0 a 7) in cui si trova quel carattere e'data da:

LINEA = Y AND 7

Il bit (punto) di quel BYTE (linea) e' dato da:

BIT = 7-(X and 7)

Riunendo queste formule, si ottiene il byte in cui e' locato il punto (X,Y) della memoria carattere:

BYTE = BASE+ROW*320+CHAR*8+LINEA

Mentre ogni bit della matrice di coordinate (X,Y) sara' attivato da:

POKE BYTE,PEEK(BYTE)OR 2 bit

Il seguente programma esemplificativo traccia una curva sinusoidale:

```
50 FORx=0TO319STEP.5:REM ONDA SINUSOIDALE
60 V=INT(90+80*SIN(X/10)
70 CH=INT(X/8)
80 RO=INT(V/8)
85 LN=VAND7
90 BY=BASE+RO*320+8*CH+LN
100=BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2 BI)
120 NEXTX
125 POKE1024,16
130 GOTO130
```

Il calcolo di cui alla linea 60 modifica i valori della funzione seno dall'intervallo (-1,1) all'intervallo (10,170). Le istruzioni da 70 100 calcolano il carattere, la riga, il byte ed il bit interessati usando le formule precedentememte illustrate. L'istruzione 125 segnala il termine del programma cambiando il colore dell'angolo in alto a sinistra dello schermo. L'istruzione 130 fa entrare il programma in un LOOP infinito. Al termine della visualizzazione, premere RUN/STOP e quindi RESTORE.
Come ulteriore esempio, si puo' modificare il programma della curva sinusoidale appena illustrato visualizzando un semicerchio. I cambiamemti necessari sono dati nelle istruzioni che seguono:

```
50 FORX=0TO160:REM DIVIDE A META' LO SCHERMO
```

```
55 Y1=100+SQR(160*X-X*X)
56 Y2=100-SQR(160*X-X*X)
60 FORY=Y1TOY2STEPY1-Y2
70 CH=INT(X/8)
80 RO=INT(Y/8)
85 LN=YAND7
90 BY=BASE+RO*320+8*CH+LN
100 BI=7-(XAND7)
110 POKEBY,PEEK(BY)OR(2 BI)
114 NEXT
```

In questo modo viene disegnato un semicerchio nell'area HI-RES dello schermo.

AVVERTENZA: Le variabili BASIC possono sovrapporrsi allo schermo ad alta risoluzione. Se c'e' bisogno di altro spazio di memoria occorre spostare la parte bassa del BASIC oltre l'area dello schermo ad alta risoluzione. Questa situazione NON si verifica in linguaggio macchina, ma SOLAMENTE in BASIC.

MODO BIT MAP MULTICOLORE

Come i caratteri del modo multicolore, il modo bit map multicolore consente di visualizzare fino a 4 differenti colori in ogni sezione 8x8 della bit map. E come nel modo multi-carattere, cio' comporta una riduzione della risoluzione orizzontale (da 320 a 160 punti).
Il modo bit map multicolore usa per la bit map una sezione di 8K di memoria. La selezione dei colori per il modo bit map multicolore avviene:

1) Dal registro 0 del colore di fondo
2) Dalla matrice video (i 4 bit piu' alti danno un colore, i 4 piu' bassi un altro)
3) Dalla memoria del colore

Il modo bit map multicolore viene attivato impostando a 1 il bit 4 della locazione 53270 ($D016 HEX), tramite la seguente POKE:

```
POKE 53265,PEEK(53265)OR32:POKE53270,PEEK(53270)OR 16
```

Analogamemte, il modo Bit Map multicolore viene disattivato impostando a 0 i due registri di cui sopra, tramite la seguente POKE:

```
POKE 53265,PEEK(53265)AND 223:POKE 53270,PEEK(53270)AND 239
```

Come nel modo bit map standard (HI-RES),si crea una corrispondenza univoca tra il segmento di 8K di memoria usato per la visualizzazione e quanto viene visualizzato sullo schermo. Tuttavia, i punti orizzontali sono larghi 2 Bit, e formano nell'area di memoria video un punto per il quale si ha a disposizione uno fra quattro colori.

BIT	PROVENIENZA DELL'INFORMAZIONE SUL COLORE
00	Colore #0 di fondo (colore schermo)
01	4 bit piu' alti della memoria schermo
10	4 bit bassi della memoria schermo
11	Semibyte colore (1 nybble=1/2byte=4bits)

SCROLLING RALLENTATO

Il circuito VIC-II consente lo scrolling rallentato sia in orizzontale che in verticale. Lo scrolling rallentato consiste nello spostamento di un pixel di tutto lo schermo in una direzione. Questo spostamento puo' avvenire verso l'alto, il basso, la destra o la sinistra. Si usa per fare apparire lentamente sullo schermo nuove informazioni mentre quelle vecchie escono lentamente dalla parte opposta.

Anche se il circuito VIC-II svolge gran parte di questo lavoro, lo scrolling effettivo deve essere effettuato da un programma in linguaggio macchina. Il circuito VIC-II possiede la capacita' di porre lo schermo video in una qualunque di 8 posizioni orizzontali ed 8 verticali. Questo posizionamento e' controllato dai registri di scrolling del VIC-II. Questo circuito possiede anche un modo a 24 righe e 38 colonne; queste misure ridotte dello schermo vengono utilizzate per fare posto ai nuovi dati dai quali deve partire lo scrolling.

Lo scrolling rallentato può essere riassunto nei seguenti punti:

1) Stringere lo schermo (il bordo, di conseguenza, si espande);
2) Impostare il registro di scrolling al valore massimo o minimo (secondo la direzione in cui si vuole eseguire lo scrolling);
3) Introdurre i nuovi dati nella posizione dello schermo piu'adatta (i dati non vengono visualizzati);
4) Aumentare (diminuire) il registro di scroll finche' non raggiunga il valore massimo (minimo)
5) Usare la procedura in linguaggio macchina per muovere lo schermo di un carattere nella direzione dello scroll.
6) Ritornare al punto 2).

Per portarsi nel modo a 38 colonne, occorre impostare a 0 il bit 3 della locazione 53270 ($D016 HEX) con la seguente POKE:

```
POKE 53270,PEEK(53270)AND 247
```

Per tornare nel modo a 40 colonne, impostare a 1 lo stesso bit di cui sopra:

```
POKE 53270,PEEK(53270)OR 8
```

Il modo a 24 righe viene attivato impostando a 0 il bit 3 della locazione 53265 ($D011 HEX):

```
POKE 53265,PEEK(53265)AND 247
```

Mentre il ritorno nel modo a 25 righe avverra' impostando a 1 il bit

precedente:

POKE 53265,PEEK(53265)OR 8

Eseguendo uno scrolling nella direzione X (righe), occorre porre il circuito VIC-II nel modo a 38 colonne, in modo di fare posto ai nuovi dati da cui iniziare lo scroll. Se lo scrolling avviene verso SINISTRA, i nuovi dati vanno piazzati sulla DESTRA, e viceversa. Da notare che la memoria dello schermo ha ancora 40 colonne, ma solamente 38 sono visibili.
Eseguendo uno scrolling nella direzioneY (colonne), occorre porre il circuito VIC-II nel modo a 24 righe. Se lo scrolling avviene verso l'ALTO, i dati vanno piazzati nell'ULTIMA riga,e viceversa. Diversamente dallo scrolling lungo X, dove c'e' un'area non visualizzata da ogni lato dello schermo, nello scrolling lungo Y c'e' solamente un'area non visualizzata. Quando il registro di scrolling nella direzione Y viene impostato a 0, non viene visualizzata la prima linea, che in questo modo viene messa a disposizione dei nuovi dati. Quando invece il registro di scrolling nella direzione Y viene impostato a 7, non viene visualizzata l'ultima riga.
Per eseguire lo scrolling nella direzione X, il registro di scroll e'locato nei bit da 2 a 0 della locazione 53270 ($D016 HEX), in cui si trova il registro di controllo del VIC-II. Come sempre,e'importante modificare solamente questi bit; cio' si ottiene con la seguente POKE:

POKE 53270,(PEEK(53270)AND248)+X

Dove X e' un numero compreso tra 0 e 7 che indica la posizione X dello schermo. Per eseguire lo scrolling nella direzione Y, il registro di scroll e' locato nei bit da 2 a 0 della locazione 53265 ($D011 HEX),in cui si trova il registro di controllo del VIC-II. Per modificare solamente questi bit si usa la seguente POKE:

POKE 53265,(PEEK(53265)AND 248)+Y

Dove Y e' un numero compreso fra 0 e 7 che indica la posizione Y dello schermo. Per eseguire lo scroll di un testo partendo dal basso, impostare da 0 a 7 i tre bit piu' bassi della locazione 53265, introdurre altri dati nella linea non visualizzata nella parte bassa dello schermo, e poi ripetere il passaggio. Per eseguire lo scroll da sinistra a destra, impostare da 0 a 7 i tre bit piu' bassi della locazione 53270, riportare una nuova colonna di dati sulla colonna 0 dello schermo, e quindi ripetere il passaggio.
Impostando i bit di scroll a partire da 1, il testo si muove nella direzione opposta.

ESEMPIO – Scrolling eseguito partendo dal basso dello schermo

```
10 POKE53265,PEEK(53265)AND247       :REM PASSA AL MODO 24 RIGHE
20 PRINTCHR$(147)                    :REM AZZERA LO SCHERMO
30 FORX=1TO24:PRINTCHR$(17);:NEXT    :REM POSIZIONA IL CURSORE IN BASSO
40 POKE53265,(PEEK(53265)AND248)+7  :REM SI POSIZIONA PER IL PRIMO
41 REM                                    "SCROLL"
50 PRINT"     HELLO";
60 FORP=6OTO0STEP-1
70 POKE53265,(PEEK(53265)AND248)+P
80 FORX=1TO50:NEXT                   :REM CICLO DI RITARDO
90 NEXT:GOTO40
```

ANIMAZIONI

Un'animazione e' un tipo di carattere speciale, definibile dall'utente, che puo' essere visualizzato dovunque sullo schermo Tutto cio' che si deve fare e' comunicare ad un'animazione "a cosa somigliare", "di che colore essere", e "dove comparire", il resto lo fa il circuito VIC-II! Le animazioni possono essere in uno qualunque dei 16 colori disponibili.

Le animazioni possono essere usate con QUALUNQUE modo della grafica, sia bit map, carattere, multicolore, ecc., mantenendo la propria forma in ognuno. L'animazione possiede un colore, un modo (HI-RES o multicolore) ed una forma propri.

Il circuito VIC-II puo' conservare automaticamente, ad un certo istante, fino a 8 animazioni. Un numero maggiore puo' essere visualizzato usando la tecnica di INTERRUZIONE DEL QUADRO.

Caratteristiche delle animazioni:

1) Dimensioni: 24 punti orizzontali X 21 verticali.
2) Controllo di colore indivuduale per ogni animazione.
3) Modo animazione multicolore.
4) Ingrandimento (2X) orizzontale, verticale o entrambi.
5) Priorita' selezionabile tra animazione e fondo.
6) Priorita' fisse tra animazioni.
7) Individuazione dei punti di contatto tra animazioni.
8) Individuazione dei punti di contatto tra animazionie e fondo.

Queste particolari capacita' di un'animazione semplificano la stesura di molti giochi. Poiche' le animazioni sono conservate dall'hardware, e' persino possibile scrivere giochi in BASIC di buona qualita'. Il circuito VIC-II sostiene direttamente fino a 8 animazioni, numerate da 0 a 7. Ogni animazione ha la propria locazione di definizione, i propri registri di posizione e colore ed i propri bit per l'abilitazione e la scoperta dei punti di contatto.

DEFINIZIONE DI UN'ANIMAZIONE

Le animazioni vengono definite come i caratteri programmabili, solo che, essendo di dimensioni maggiori, richiedono una maggiore quantita' di byte. Come gia' detto, un'animazione e' composta da 24 punti X 21, per un totale di 504 punti, pari a 63 bytes (504/8 bit), questi ultimi diposti su 21 righe di 3 byte ciascuna. Una definizione di animazione puo' essere data come segue:

```
BYTE 0     BYTE 1     BYTE 2
BYTE 3     BYTE 4     BYTE 5
BYTE 6     BYTE 7     BYTE 8
  ..         ..         ..
  ..         ..         ..
  ..         ..         ..
BYTE 60    BYTE 61    BYTE 62
```

Un altro modo per la creazione di un'animazione e' considerare il blocco di definizione di una'animazione a livello di bit, come illustrato nella figura 3.2.

COLONNA	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
BIT	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
VALORE (ON = 1×VAL)	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
RIGA 0																								
RIGA 1																								
RIGA 2																								
RIGA 3																								
RIGA 4																								
RIGA 5																								
RIGA 6																								
RIGA 7																								
RIGA 8																								
RIGA 9																								
RIGA 10																								
RIGA 11																								
RIGA 12																								
RIGA 13																								
RIGA 14																								
RIGA 15																								
RIGA 16																								
RIGA 17																								
RIGA 18																								
RIGA 19																								

Figura 3.2 - Blocco per la definizione di un'animazione

In un'animazione standard (HI-RES), ogni bit impostato a 1 viene visualizzato nel colore principale di quell'animazione, mentre ogni bit impostato a 0 e' trasparente e visualizza qualunque dato ci sia dietro di esso, analogamente ad un carattere standard.
Le animazioni multicolore sono simili ai caratteri multicolore. La perdita di risoluzione orizzontale e' bilanciata dalla particolare risoluzione del colore. La risoluzione dell'animazione comprende 12 punti orizzontali X 21 verticali. Ogni punto dell'animazione ha una grandezza doppia, ma il numero di colori visualizzati nell'animazione sale a 4.

PUNTATORI DELL'ANIMAZIONE

Oltre ai 63 byte necessari per definire ogni animazione, e' necessario un altro byte contenente uno spazio alla fine di ogni animazione. Quindi ognuna di esse occupa 64 byte, il che semplifica la ricerca in memoria della definizione dell'animazione, poiche' 64 e' un numero pari e, in binario, una potenza pari.
Ciascuna delle 8 animazioni ha un byte associato chiamato PUNTATORE ALL'ANIMAZIONE. Questi puntatori controllano la locazione in memoria di ogni definizione di animazione. Sono sempre locati negli ultimi 8 byte del segmento di 1K della memoria dello schermo; cio' significa che, sul COMMODORE 64, la loro locazione inizia di solito a 2040 ($07F8 HEX). Tuttavia, questa locazione segue gli spostamenti assegnati alla memoria video.
Ogni puntatore all'animazione contiene un numero compreso fra 0 e 255, che punta alla definizione di quell'animazione. Poiche' ogni definizione di animazione occupa 64 byte, il puntatore puo' "vedere" dovunque nel blocco di memoria di 16 K a cui puo' accedere il circuito VIC-II (256*64 = 16K).
Se il puntatore #0 dell'animazione, situato alla locazione 2040, contiene ad esempio il numero 14, cio' vuol dire che l'animazione 0 sara' visualizzata usando i 64 bytes che iniziano alla locazione 14 X 64 = 896, situata nel buffer della cassetta. Quindi:

LOCAZIONE=(BANCO * 16384)+(VALORE DEL PUNTATORE DELL'ANIMAZIONE * 64)

Essendo BANCO il segmento di memoria di 16K accessibile dal circuito VIC-II a quell'istante; il suo valore e' compreso fra 0 e 3. Questa formula fornisce il byte di partenza del blocco di 64 byte di definizione dell'animazione.
Quando il circuito VIC-II sta accedendo al BANCO 0 o al BANCO 2, come ricordato in precedenza, in alcune locazioni ci possono essere configurazioni ROM dell'insieme carattere. In queste locazioni NON ci devono essere definizioni di animazioni. Se per qualche motivo si ha bisogno di piu' di 128 differenti definizioni di animazioni, si deve ricorrere ai banchi 1 o 3 che sono sprovvisti di CONFIGURAZIONI ROM

ATTIVAZIONE DELLE ANIMAZIONI

Il registro di controllo del VIC-II, locato a 53269 ($D015 HEX), e' conosciuto come registro di ABILITAZIONE ANIMAZIONE. Questo registro contiene un bit di ciascuna animazione, in modo da poter controllare se una data animazione sia stata attivata oppure no. La forma del registro e' la seguente:

$D015 76543210

Ad esempio, per attivare l'animazione 1 e' necessario impostare a 1 il corrispondente bit:

```
POKE53269,PEEK(53269)OR 2
```

Quindi, generalizzando:

```
POKE 53269,PEEK(53269)OR (2↑SN)
```

Essendo SN il numero dell'animazione; quest'ultimo deve essere compreso fra 0 e 7.

NOTA: Un'animazione deve essere attivata (ON) prima di diventare visibile.

DISATTIVAZIONE DELLE ANIMAZIONI

Un'animazione viene disattivata impostando a 0 il corrispondente bit del registro di controllo del VIC-II locato a 53269 ($D015 HEX):

```
POKE 53269, PEEK(53269)AND (255-2↑SN)
```

Essendo SN il numero di attivazione; quest'ultimo deve essere compreso fra 0 e 7.

COLORI

Un'animazione puo' assumere uno qualunque dei 16 colori generati dal circuito VIC-II. Ciascuna animazione ha il proprio registro colore, le cui locazioni di memoria sono:

INDIRIZZO	DESCRIZIONE
53287 ($D027)	REGISTRO COLORE DELL'ANIMAZIONE 0
53288 ($D028)	REGISTRO COLORE DELL'ANIMAZIONE 1
53289 ($D029)	REGISTRO COLORE DELL'ANIMAZIONE 2
53290 ($D02A)	REGISTRO COLORE DELL'ANIMAZIONE 3
53291 ($D02B)	REGISTRO COLORE DELL'ANIMAZIONE 4
53292 ($D02C)	REGISTRO COLORE DELL'ANIMAZIONE 5
53293 ($D02D)	REGISTRO COLORE DELL'ANIMAZIONE 6
53294 ($D02E)	REGISTRO COLORE DELL'ANIMAZIONE 7

Tutti i punti accesi dell'animazione sono visualizzati con il colore contenuto nel registro colore dell'animazione. La rimanente parte dell'animazione e' trasparente, cioe' nei punti "spenti" viene visualizzato qualunque cosa si trovi dietro l'animazione (generalmente lo spazio, locato nel REGISTRO COLORE DELL'ANIMAZIONE 0).

MODO MULTICOLORE

Il modo multicolore mette a disposizione di ogni animazione fino a 4 differenti colori. Tuttavia, come negli altri modi multicolore, la risoluzione orizzontale e' dimezzata: in altri termini, quando si lavora nel modo multicolore dell'animazione, come nel modo multicolore carattere, anziche' avere a disposizione per tutta l'animazione 24 punti, si hanno 12 coppie di punti, ognuna delle quali viene chiamata

coppia di bit (coppia di punti) come un singolo punto dell'intera animazione. La seguente tabella fornisce i valori della coppia di bit necessari all'attivazione di ciascuno dei quattro colori scelti per quell'animazione.

COPPIE DI BIT	DESCRIZIONE
00	COLORE VIDEO TRASPARENTE
01	REGISTRO ANIMAZIONE MULTICOLORE #0 (53285) ($D025)
10	REGISTRO COLORE DELL'ANIMAZIONE
11	REGISTRO ANIMAZIONE MULTICOLORE #1 (53286) ($D026)

IMPOSTAZIONE DI UN'ANIMAZIONE NEL MODO MULTICOLORE

Per attivare un'animazione nel modo multicolore, occorre impostare a ON il registro di controllo del VIC-II locato a 53276 ($D01C HEX):

POKE 53276,PEEK(53276) OR (2↑SN)

dove SN e' il numero di animazione, che deve essere compreso fra 0 e 7. Per disattivare un'animazione dal modo multicolore, occorre impostare a OFF lo stesso registro precedente:

POKE 53276,PEEK(53276) AND (255−2↑SN)

dove SN e' il numero di animazione (compreso fra 0 e 7).

ANIMAZIONI INGRANDITE

Il circuito VIC-II ha la capacita' di ingrandire un'animazione in verticale, in orizzontale o in entrambe le direzioni. Ovviamente, ogni punto ingrandito ha le sue dimensioni raddoppiate, senza alcun aumento della risoluzione...semplicemente, l'animazione diventa piu' grande.
L'ingrandimento orizzontale di un'animazione si ottine impostando a ON (mettendo a 1) il bit corrispondente del registro di controllo del VIC-II locato a 53277 ($D01D HEX):

POKE 53277,PEEK(53277)OR (2↑SN)

dove SN e' il numero di animazione (compreso tra 0 e 7).
Per riportare in grandezza normale un'animazione ingrandita in orizzontale, occorre impostare a OFF (mettere a 1) il corrispondente bit del registro di controllo del VIC-II locato a 53277 ($01D HEX):

POKE 53277,PEEK(53277)AND (255−2↑SN)

dove SN e' il numero di animazione, che deve essere compreso fra 0 e 7.
Per espandere un'animazione nella direzione verticale, occorre impostare a ON (mettere a 1) il corrispondente bit del registro di controllo del VIC-II, locato a 53271 ($D017 HEX):

POKE 53271,PEEK(53271)OR (2↑SN)

dove SN e' il numero di animazione (compreso fra 0 e 7).
Per riportare in grandezza normale un'animazione ingrandita in verticale, occorre impostare ad OFF (mettere a 0) il corrispondente

bit del registro di controllo del VIC-II locato a 53271 ($D017 HEX):

```
POKE 53271,PEEK(53271)AND (255-2↑SN)
```

dove SN e' il numero di animazione (compreso fra 0 e 7).

POSIZIONAMENTO DELLE ANIMAZIONI

Una volta creata un'animazione, il COMMODORE 64 usa tre registri di posizionamento per muovere l'animazione sullo schermo:

1) REGISTRO POSIZIONE X DELL'ANIMAZIONE (ORIZZONTALE)
2) REGISTRO POSIZIONE Y DELL'ANIMAZIONE (VERTICALE)
3) BIT PIU' SIGNIFICATIVO DEL REGISTRO POSIZIONE X

Ogni animazione ha 512 posizioni possibili lungo l'asse X e 256 lungo l'asse Y.
I registri di posizione X e Y lavorano in coppia. Le locazioni di questi registri appaiono in memoria come segue: prima il registro X per l'animazione 0, poi il corrispondente registro Y; segue poi la coppia di registri dell'animazione 1 e cosi' via. I 16 registri X e Y sono seguiti dal bit piu' significativo della posizione X (MSB X), locato nel proprio registro.
La tabella seguente illustra la locazione del registro di posizione di ogni animazione; il loro uso avviene per mezzo di una POKE.

LOCAZIONE		DESCRIZIONE
DECIMALI	HEX	
53248	($D000)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 0
53249	($D001)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 0
53250	($D002)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 1
53251	($D003)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 1
53252	($D004)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 2
53253	($D005)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 2
53254	($D006)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 3
53255	($D007)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 3
53256	($D008)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 4
53257	($D009)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 4
53258	($D010)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 5
53259	($D011)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 5
53260	($D012)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 6
53261	($D013)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 6
53262	($D014)	REGISTRO POSIZIONE X DELL'ANIMAZIONE 7
53263	($D015)	REGISTRO POSIZIONE Y DELL'ANIMAZIONE 7
53264	($D016)	REGISTRO POSIZIONE X DELL'ANIMAZIONE X MSB

La posizione di un'animazione viene calcolata a partire dall'angolo IN ALTO A SINISTRA della zona di 24 punti X 21 in cui e' stata disegnata l'animazione. NON importa da quanti punti e' formata un'animazione. Anche se, come animazione, si usasse un solo punto, e lo si volesse visualizzare al centro dello schermo, si dovrebbe lo stesso calcolare l'esatto posizionamento a partire dalla locazione dell'angolo in alto a sinistra.

Posizionamento Verticale

L'allestimento delle posizioni orizzontali e' leggermente piu' complesso del posizionamento verticale, per cui comincermo l'esposizione da quest'ultimo.

Nella direzione verticale dello schermo TV si possono programmare indipendentemente 200 posizioni di punti; i registri di posizione Y dell'animazione possono trattare numeri fino a 255. Si ha quindi un'eccedenza nelle locazioni del registro disponibili per il movimento di un'animazione. E' anche possibile il movimento lento di un'animazione, per il quale sono a disposizione piu' di 200 valori.

Partendo dall'alto del video, il primo valore che fa comparire sul video una animazione non ingrandita e', per la direzione verticale, 30, mentre se l'animazione e' ingrandita tale valore e' 9 (diversamente, si avrebbe una perdita di risoluzione, poiche' ogni punto e' alto il doppio, e la posizione iniziale e' ancora calcolata a partire dall'angolo in alto a sinistra dell'animazione).

Il primo valore di Y per il quale un'animazione (ingrandita o no) compare interamente sullo schermo (vengono visualizzate tutte le 21 linee disponibili) e' 50. L'ultimo valore di Y per il quale un'animazione non ingrandita compare interamente sullo schermo e' 229; se l'animazione e' ingrandita, tale valore si riduce a 208. Il primo valore di Y per il quale un'animazione e' completamente fuori schermo e' 250.

ESEMPIO:

```
      SHIFT CLR/HOME
10 PRINT"♥" : REM AZZERA LO SCHERMO
20 POKE2040,13 : REM LEGGE DAL BLOCCO 13 I DATI DELL'ANIMAZIONE 0
30 FORI=0TO62:POKE832+I,129:NEXT : REM INSERISCE NEL BLOCCO 13
31 REM I DATI DELL'ANIMAZIONE (13*64=832)
40 V=53248! : REM SI DISPONE ALL'INIZIO DEL CIRCUITO VIDEO
50 POKEV+21,1 : REM ABILITA L'ANIMAZIONE 1
60 POKEV+39,1 : REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
70 POKEV+1,100 : REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
80 POKEV+16,0:POKEV,100 : REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
```

POSIZIONE ORIZZONTALE

Il posizionamento nella direzione orizzontale e' piu' complesso, poiche' ci sono 256 posizioni, il che comporta l'introduzione di un bit speciale, il nono, usato per controllare la posizione lungo l'asse X. Aggiungendo, quando necessario, questo bit, un'animazione viene ad avere a disposizione 512 possibili posizioni per il movimento destra/sinistra nella direzione X. Si ha cosi' un aumento delle possibili combinazioni che possono essere visualizzate sulla parte visibile dello schermo. Anche se ogni animazione puo' assumere una posizione compresa fra 0 e 511, lo schermo consente la visualizzazione dei valori compresi tra 24 e 343. Se la posizione X di un'animazione e' maggiore di 255 (sulla destra dello schermo), il bit del registro POSIZIONE DEL BIT PIU' SIGNIFICATIVO nella posizione X deve essere impostato a 1. Se la posizione X di un'animazione e' minore di 256 (sulla sinistra dello schermo), allora l'MSB nella direzione X di quell'animazione deve essere 0. I bit da 0 a 7 del registro MSB nella direzione X corrispondono rispettivamente alle animazioni 0 a 7.

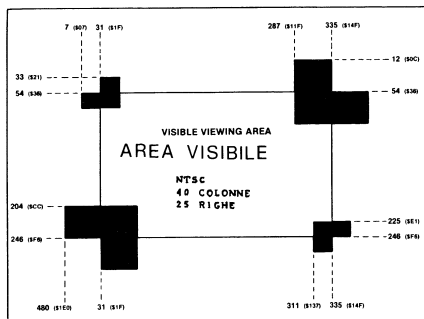
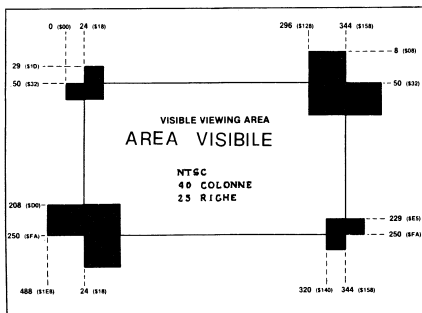



Figura 3.3 - Tabelle per il posizionamento delle animazioni

ESEMPIO:

```
10 PRINT"[SHIFT CLR HOME]"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV+1,100
80 FORJ=0TO347
90 HX=INT(J*256):LX=J-256*HX
100 POKE,LX:POKEV,LX:POKEV+16,HX:NEXT
```

Quando si muovono animazioni ingrandite verso la sinistra dello schermo nella direzione X, occorre far partire l'animazione DA FUORI SCHERMO sulla DESTRA, in quanto un'animazione ingrandita e' piu' grande della quantita' di spazio disponibile sulla sinistra dello schermo.

ESEMPIO:

```
10 PRINT"[SHIFT CLR HOME]"
20 POKE2040,13
30 FORI=0TO62:POKE832+I,129:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1:POKEV+23,1:POKEV+29,1
70 POKEV+1,100
80 J=488
90 HX=INT(J*256):LX=J-256*HX
100 POKEV,LX:POKEV+16,HX
110 J=J+1:IFJ<511THENJ=0
120 IFJ>488ORJ<348GOTO90
```

Per il posizionamento di una animazione si veda la figura 3.3.
L'uso di questi valori consente di posizionare un'animazione dovunque. E' facile realizzare un movimento molto lento spostando l'animazione di un solo punto alla volta.

RIASSUNTO DEL POSIZIONAMENTO DELLE ANIMAZIONI

Animazioni non ingrandite sono visibili, almeno parzialmente, nel modo a 40 colonne X 25 riche, impostando:

$$1 <= X <= 343$$

$$30 <= Y <= 249$$

Nel modo a 38 colonne, il parametro X assume

$$8 <= X <= 334$$

Nel modo a 24 righe, il parametro Y assume

$$34 <= Y <= 245$$

Animazioni ingrandite sono visibili, almeno parzialmente, nel modo a

40 colonne X 25 righe, impostando

489 > = X < = 343

9 > = Y < = 249

Nel modo a 38 colonne, il parametro X assume

496 > = X < = 334

Nel modo a 24 righe, il parametro Y assume

13 < = Y < = 245

PRIORITÀ DI VISUALIZZAZIONE DELLE ANIMAZIONI

Le animazioni possono incrociarsi, come pure passare davanti o dietro ad altri oggetti presenti sullo schermo. Si ha cosi' un vero effetto tridimensionale per i giochi.

La priorita' tra animazioni e' fissa: la piu' alta e' quella dell'animazione 0, la piu' bassa e' quella dell'animazione 7. Se ad esempio si incrociano le animazioni 1 e 6, la 1 passera' davanti alla 6.

Percio', quando si stabilisce quali figure far apparire in primo piano, occorre assegnare loro numeri di animazione piu' bassi di quelli delle animazioni in secondo piano.

NOTA: E' possibile un effetto "finestra": se un'animazione ha una priorita' maggiore di un'altra, ed ha al suo interno dei "buchi" (zone di punti non impostati a 1 e quindi spenti), allora l'animazione di priorita' piu' bassa sara' visibile attraverso tali "buchi". Cio' avviene anche fra un'animazione e un dato che compare sullo sfondo.

La priorita' tra le animazioni e lo sfondo e' controllabile per mezzo del registro di priorita' ANIMAZIONE-SFONDO posto nella locazione 53275 ($D01B HEX). Questo registro contiene un bit per ogni animazione: se questo bit vale 0, l'animazione ha una priorita' maggiore dello sfondo, cioe' compare davanti ai dati presenti sullo sfondo; se invece il bit vale 1, l'animazione ha una priorita' minore dello sfondo, per cui l'animazione compare dietro ai dati presenti sullo sfondo.

DETERMINAZIONE DEI PUNTI DI CONTATTO

Una delle caratteristiche piu' interessanti del circuito del VIC-II e' la capacita' di scoprire i punti di contatto tra le animazioni o tra le animazioni e i dati che compaiono sullo sfondo. Un contatto avviene quando una parte "non zero" (i cui bit sono impostati a 1, visualizzando quindi quei particolari punti) di un'animazione si sovrappone ad una parte "non zero" di un'altra animazione o di un carattere.

PUNTI DI CONTATTO TRA ANIMAZIONI

I punti di contatto fra animazioni vengono riconosciuti dal computer e segnalati nel registro contatto tra animazioni locato a 53278 ($D01E

HEX) del registro di controllo del circuito VIC-II.
Il registro contatto fra animazioni contiene un bit per ogni animazione. Se tale bit e' a 1, cio' vuol dire che la corrispondente animazione e' interessata in un contatto. I bit di questo registro rimangono impostati fino alla successiva lettura (che avviene tramite PEEK). Una volta letto, il registro viene automaticamente azzerato, per cui e' consigliabile salvare il contenuto in una variabile per tutto il tempo che se ne ha bisogno.

NOTA: I contatti possono avvenire anche fra animazioni fuori schermo

Questo punto di contatto viene riconosciuto dal registro di contatto ANIMAZIONI-DATI locato a 53279 ($D01F HEX) del registro di controllo del circuito VIC-II.
Il registro di contatto ANIMAZIONE-DATI contiene un bit per ogni animazione: se questo e' a 1, allora la corrispondente animazione e' interessata in un contatto. I bit di questo registro rimangono impostati fino alla successiva lettura (che avviene tramite PEEK). Una volta letto, il registro viene azzerato automaticamente, per cui e' di nuovo consigliabile il salvataggio del contenuto in una variabile.

NOTA: Il dato multicolore 01 e' considerato trasparente al contatto, anche se viene visualizzato. Quando si imposta uno schermo di fondo e' consigliabile procedere in modo da evitare un contatto con 01 nel modo multicolore.

```
10 REM ANIMAZIONI - ESEMPIO 1
20 REM LA MONGOLFIERA
30 VIC=13*4096:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,1:REM ABILITA L'ANIMAZIONE 0
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,1:REM INGRANDISCE L'ANIMAZIONE 0 LUNGO LA DIREZIONE Y
38 POKEVIC+29,1:REM INGRANDISCE L'ANIMAZIONE 0 LUNGO LA DIREZIONE X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
220 POKEVIC+39,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
250 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
251 REM             UN'ANIMAZIONE
300 READA:REM :LEGGE UN BYTE
310 POKE192*64+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXTY:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIC):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO DELLO...
370 REM SCHERMO, ALLORA INVERTE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=1
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM SOMMA DELTA X A X
470 X=XAND255 REM CONTROLLA CHE X SIA COMPRESO NEI LIMITI CONSENTITI
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INSERISCE IL NUOVO VALORE DI X NELLA POSIZIONE X
491 REM            DELL'ANIMAZIONE 0
510 POKEVIC+1,Y:REM INSERISCE IL NUOVO VALORE DI Y NELLA POSIZIONE Y
511 REM              DELL'ANIMAZIONE 0
530 GOTO40
600 REM ***** DATI ANIMAZIONE *****
610 DATA0,127,0,1,255,192,3,255,224,3,231,224
620 DATA7,217,240,7,220,240,7,217,240,3,231,224
630 DATA3,255,224,3,255,224,2,255,160,1,127,64
640 DATA1,62,64,0,156,128,0,156,128,0,73,0,0,73,0,
650 DATA0,62,0,0,62,0,0,62,0,0,28,0,0
```

```
10 REM ANIMAZIONI - ESEMPIO 2
20 REM ANCORA LA MONGOLFIERA
30 VIC=13*4096:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,63:REM ABILITA LE ANIMAZIONI 0-5
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,3:REM INGRANDISCE LE ANIMAZIONI 0 E 1 LUNGO Y
38 POKEVIC+29,3:REM INGRANDISCE LE ANIMAZIONI 0 E 1 LUNGO X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
50 POKE2041,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 1
60 POKE2042,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 2
70 POKE2043,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 3
80 POKE2044,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 4
90 POKE2045,193:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 5
100 POKEVIC+4,30:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 2
110 POKEVIC+5,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 2
120 POKEVIC+6,65:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 3
130 POKEVIC+7,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 3
140 POKEVIC+8,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 4
150 POKEVIC+9,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 4
160 POKEVIC+10,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 5
170 POKEVIC+11,58:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 5
175 PRINT"[CTRL 2][SHIFT CLR HOME]"TAB(15)"QUESTE SONO DUE ANIMAZIONI HI-RES"
176 PRINTTAB(55)"UNA SOPRA L'ALTRA"
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
200 POKEVIC+2,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 1
210 POKEVIC+3,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 1
220 POKEVIC+39,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
230 POKEVIC+41,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 2
240 POKEVIC+43,1:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 4
250 POKEVIC+40,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 1
260 POKEVIC+42,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 3
270 POKEVIC+44,6:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 5
280 FORX=192TO193:REM INIZIO DEL CICLO DI DEFINIZIONE
281 REM                 DELLE ANIMAZIONI
290 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
291 REM              UN'ANIMAZIONE
300 READA:REM LEGGE UN BYTE
310 POKEX*64+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXTY,X:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIV):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO...
370 REM DELLO SCHERMO, ALLORA INVERTE DELTA Y
380 IFX=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDDX=1THENX=-1:SIDE=3
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM ADD DELTA X TO X
470 X=XAND255:REM CONTROLLA CHE X SIA NEI LIMITI CONSENTITI
```

```
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE X
491 REM             DELL'ANIMAZIONE 0
500 POKEVIC+2,X:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE X
501 REM             DELL'ANIMAZIONE 1
510 POKEVIC+1,Y:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE Y
511 REM             DELL'ANIMAZIONE 0
520 POKEVIC+3,Y:REM INSERISCE IL NUOVO VALORE NELLA POSIZIONE Y
521 REM             DELL'ANIMAZIONE 1
530 GOTO340
600 REM ***** DATI ANIMAZIONI *****
610 DATA0,255,0,3,153,192,7,24,224,7,56,224,14,126,112,14,126,112,14,
    126,112
620 DATA6,126,96,7,56,224,7,56,224,1,56,128,0,153,0,0,90,0,0,56,0
630 DATA0,56,0,0,0,0,0,0,0,0,126,0,0,42,0,0,840,0,40,0,0
640 DATA0,0,0,0,102,0,0,231,0,0,195,0,1,129,128,1,129,128,1,129,128
650 DATA1,129,128,0,195,0,0,195,0,4,195,32,2,102,64,2,36,64,1,0,128
660 DATA1,0,128,0,153,0,0,153,0,0,0,0,0,84,0,0,42,0,0,20,0,0
```

```
10 REM ANIMAZIONI - ESEMPIO 3
20 REM THE HOT AIR GORF
30 VIC=53248:REM LOCAZIONE DI INIZIO DEI REGISTRI DEL VIC
35 POKEVIC+21,1:REM ABILITA L'ANIMAZIONE 0
36 POKEVIC+33,14:REM IMPOSTA IL COLORE DI FONDO A BLU CHIARO
37 POKEVIC+23,1:REM INGRANDISCE L'ANIMAZIONE 0 NELLA DIREZIONE Y
38 POKEVIC+29,1:REM INGRANDISCE L'ANIMAZIONE 0 NELLA DIREZIONE X
40 POKE2040,192:REM IMPOSTA IL PUNTATORE DELL'ANIMAZIONE 0
50 POKEVIC+28,1:REM ATTIVA IL MODO MULTICOLORE
60 POKEVIC+37,7:REM IMPOSTA IL MULTICOLORE 0
70 POKEVIC+38,4:REM IMPOSTA IL MULTICOLORE 1
180 POKEVIC+0,100:REM IMPOSTA LA POSIZIONE X DELL'ANIMAZIONE 0
190 POKEVIC+1,100:REM IMPOSTA LA POSIZIONE Y DELL'ANIMAZIONE 0
220 POKEVIC+39,2:REM IMPOSTA IL COLORE DELL'ANIMAZIONE 0
290 FORY=0TO63:REM CONTATORE BYTE CON CICLO DI COSTRUZIONE DI
291 REM              UN'ANIMAZIONE
300 READA:REM LEGGE UN BYTE
310 POKE12288+Y,A:REM MEMORIZZA I DATI NELL'AREA ANIMAZIONE
320 NEXT Y:REM CHIUDE IL CICLO
330 DX=1:DY=1
340 X=PEEK(VIC):REM CONSIDERA LA POSIZIONE X DELL'ANIMAZIONE 0
350 Y=PEEK(VIC+1):REM CONSIDERA LA POSIZIONE Y DELL'ANIMAZIONE 0
360 IFY=50ORY=208THENDY=-DY:REM SE Y SI TROVA SUL BORDO...
370 REM DELLO SCHERMO, ALLORA INVERTE DELTA Y
380 IF X=24AND(PEEK(VIC+16)AND1)=0THENDX=-DX:REM SE L'ANIMAZIONE...
390 REM TOCCA IL BORDO SINISTRO, ALLORA LA INVERTE
400 IFX=40AND(PEEK(VIC+16)AND1)=1THENDX=-DX:REM SE L'ANIMAZIONE...
410 REM TOCCA IL BORDO DESTRO, ALLORA LA INVERTE
420 IFX=255ANDX=1THENX=-1:SIDE=1
430 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
440 IFX=0ANDDX=-1THENX=256:SIDE=0
450 REM SI POSIZIONA SULL'ALTRO LATO DELLO SCHERMO
460 X=X+DX:REM SOMMA DELTA X A X
470 X=XAND255:REM SI ASSICURA CHE X SIA COMPRESO NELL'INTERVALLO
471 REM          CONSENTITO
480 Y=Y+DY:REM SOMMA DELTA Y A Y
485 POKEVIC+16,SIDE
490 POKEVIC,X:REM INTRODUCE IL NUOVO VALORE DI X NELLA POSIZIONE X
491 REM            DELL'ANIMAZIONE 0
510 POKEVIC+1,Y:REM INTRODUCE IL NUOVO VALORE DI Y NELLA POSIZIONE Y
511 REM              DELL'ANIMAZIONE 0
520 GETA$:REM LEGGE UN CARATTERE PROVENIENTE DALLA TASTIERA
521 IFA$="M"THENPOKEVIC+28,1:REM MULTICOLORE SELEZIONATO DALL'UTENTE
522 IFA$="H"THENPOKEVIC+28,0:REM ALTA RISOLUZIONE SELEZIONATA
523 REM                            DALL'UTENTE
530 GOTO340
600 REM ***** DATI ANIMAZIONE *****
610 DATA64,0,1,16,170,4,6,170,144,10,170,160,42,170,168,41,105,104,
169,235,106
620 DATA169,235,106,169,235,106,170,170,170,170,170,170,170,170,
170,170,170,170
630 166,170,154,169,85,106,170,185,170,42,170,168,10,170,160,1,0,
64,1,0,64
640 DATA5,0,80,0
```

ALTRE CARATTERISTICHE DELLA GRAFICA

AZZERAMENTO DELLO SCHERMO

Il bit 4 del registro di controllo del Vic-II presiede la funzione di azzeramento dello schermo. Tale bit si trova nel registro di controllo alla locazione 53265 ($D011 HEX). Quando viene impostato a 1, lo schermo e' nello stato normale, quando e' impostato a 0 (disattivato) l'intero schermo assume il colore del bordo.
L'azzeramento dello schermo non comporta la perdita dei dati,semplicemente questi ultimi non vengono piu' visualizzati. Per azzerare lo schermo si usi la seguente POKE:

```
POKE 53265,PEEK(53265)AND 239
```

Mentre il ritorno dello schermo alla posizione iniziale e'dato da:

```
POKE 53265,PEEK(53265)OR 16
```

NOTA: Disabilitare lo schermo rende il processore leggermente piu' veloce. Di conseguenza, anche il programma attualmente residente in memoria acquista velocita'.

REGISTRO DI QUADRO (Televisivo)

Il registro di quadro si trova nel circuito VIC-II alla locazione 53266 ($D012 HEX), ed ha un doppio scopo. La lettura di questo registro ritorna gli 8 bit piu' bassi della attuale posizione del quadro.
La posizione di quadro del bit piu' significativo e' contenuta nel registro locato in 53265 ($D011 Hex). Il registro di quadro si usa per impostare temporanei cambiamenti del video, in modo da liberarsi dallo sfarfallio (tremolio) delle immagini sullo schermo. Questi cambiamenti devono essere eseguiti quando il quadro non si trova nella zona visibile del video, in cui le posizioni dei punti sono comprese fra 51 e 251.
Quando si scrive nel registro di quadro (compreso l'MSB), il numero scritto viene salvato per poter essere usato con la funzione di paragone del quadro. Quando il valore attuale di quest'ultimo uguaglia il numero scritto nel registro di quadro, viene impostato a 1 il bit del registro interruzione del circuito VIC-II la cui locazione e' 53273 ($D019 HEX).

NOTA: Attivando il bit appropriato, si genera un'interruzione(IRQ)

REGISTRO DI STATO DELL'INTERRUZIONE

Il registro di stato dell'interruzione contiene lo stato attuale di ogni sorgente di interruzione. Lo stato attuale del bit 2 del registro interruzione viene impostato a 1 quando si scontrano due animazioni; un analogo discorso vale anche peri bit 0...3 illustrati di seguito. Il bit 7 viene impostato a 1 qualunque interruzione avvenga. Il registro di stato dell'interruzione si trova nella locazione 53273

($D019 HEX), ed ha la seguente configurazione:

CIRCUITO	BIT	DESCRIZIONE
IRST	0	Impostato quando il contatore di quadro corrente e' uguale al contatore di quadro registrato
IMDC	1	Impostato da un contatto ANIMAZIONE-DATI (solo per il primo fino al ripristino successivo)
IMCC	2	Impostato da un contatto ANIMAZIONE-ANIMAZIONE (solo per il primo fino al prossimo ripristino)
ILP	3	Impostato dalla transizione negativa della penna ottica(uno per inquadratura)
IRQ	7	Impostato dall'attivazione di un circuito latch e abilitato

Una volta impostato, un bit di interruzione viene trascritto nell'apposito circuito "latch"; al momento del suo utilizzo, occorre azzerare tale bit impostandolo a 1. Cio' consente un trattamento selettivo dell'interruzione, senza il bisogno di registrare gli altri bit di interruzione.

Il REGISTRO ABILITATORE DELL'INTERRUZIONE e' locato a 53274 ($D01A HEX), ed ha la stessa forma del registro di stato dell'interruzione. Per tutto il tempo in cui il corrispondente bit del registro abilitatore dell'interruzione rimane impostato a 1, da questa sorgente non puo' derivare alcuna interruzione. Il registro di stato dell'interruzione puo' essere ancora consultato per ottenere delle informazioni, ma non genera piu' altre interruzioni.

Per abilitare una richiesta di interruzione il corrispondente bit di abilitazione dell'interruzione (come illustrato nella precedente tabella) deve essere impostato a 1.

Questa struttura dell'interruzione consente di usare modi di schermo distinti. Ad esempio, si puo' avere meta' schermo nel modo bit map, meta' testo, piu' di otto animazioni contemporanee, ecc. Tutto sta nell'uso corretto delle interruzioni. Per esempio, se si vuole che la parte alta dello schermo sia nel modo bit map e la parte bassa nel modo testo, occorre impostare il registro di confronto del quadro in modo che divida a meta' lo schermo (come esposto in precedenza). Al verificarsi dell'interruzione, impostare il circuito VIC-II in modo che prelievi caratteri dalla ROM, poi impostare il registro di confronto del quadro per generare un'interruzione quando si trova nella parte alta dello schermo. Al verificarsi di questa nuova interruzione, impostare il circuito VIC-II in modo che prelievi caratteri dalla RAM (modo bit map).

Si possono anche visualizzare piu' di 8 animazioni nello stesso tempo, anche se il BASIC non e' abbastanza veloce per gestire bene questa situazione; percio', se si vogliono usare le interruzioni del video, e' consigliabile lavorare in linguaggio macchina.

COMBINAZIONI CONSIGLIATE DEI COLORI DI SCHERMO E CARATTERE

I TV COLOR presentano limitazioni nell'accostamento di alcuni colori sulla stessa riga, producendo immagini confuse. La seguente tabella indica quali combinazioni colore evitare, e quali invece risaltano di piu'.

COLORE CARATTERE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	#	*	#	*	*	@	#	*	*	#	*	*	*	*	*	*
1	*	#	*	#	*	*	*	#	@	*	@	*	*	#	*	*
2	#	*	#	#	@	#	#	*	*	#	*	#	#	#	#	@
3	*	#	#	#	#	@	*	#	#	#	#	@	#	#	@	#
4	*	@	#	#	#	#	#	#	#	#	#	#	#	#	#	@
5	*	@	#	@	#	#	#	#	#	#	#	@	#	*	#	@
6	@	*	#	*	#	#	#	#	#	#	#	#	#	@	*	*
7	*	#	*	#	#	#	@	#	@	*	@	*	*	#	#	#
8	@	*	*	#	#	#	#	*	#	*	#	#	#	#	#	@
9	#	*	#	#	#	#	#	*	*	#	*	#	#	#	#	*
10	@	@	*	#	#	#	#	@	#	*	#	#	#	#	#	@
11	*	*	#	@	#	#	#	*	#	#	#	#	*	*	@	*
12	*	*	@	#	#	#	@	#	#	@	#	*	#	#	#	*
13	*	#	#	#	#	*	@	#	#	#	#	*	#	#	#	#
14	*	*	#	*	#	#	*	#	#	#	#	@	#	#	#	@
15	*	*	*	#	@	@	*	#	#	@	@	*	*	#	@	#

* = ECCELLENTE
@ = BUONO
# = SCARSO

PROGRAMMAZIONE DELLE ANIMAZIONI UN ULTERIORE SGUARDO

Questo paragrafo intende dare un approccio semplificato alle animazioni.

COSTRUZIONE DI ANIMAZIONI DA BASIC - UN BREVE PROGRAMMA

Ci sono almeno tre differenti tecniche di programmazione BASIC che consentono di creare sul COMMODORE 64 immagini grafiche ed animazioni: si puo' usare la grafica di sistema (cfr. Appendice B), creare dei caratteri personalizzati (vd. Definizioni di Carattere), oppure, meglio di tutto....usare la "grafica animata" di sistema. Per illustrare l'estrema semplicita' d'uso, basta osservare il seguente programma, uno dei piu' corti programmi di animazione che si possono scrivere in BASIC:

SHIFT CLR HOME

```
10 PRINT "♥"
20 POKE2040,13
30 FORS=832TO832+62:POKES,255:NEXT
40 V=53248
50 POKEV+21,1
60 POKEV+39,1
70 POKEV,24
80 POKEV+1,100
```

Questo programma contiene tutte le "componenti" principali necessarie per la realizzazione di qualunque animazione. I numeri delle istruzioni POKE sono stati presi dalla tabella per la costruzione delle animazioni. Questo programma definisce la prima animazione - Animazione 0 - come un cubo bianco; la descrizione del programma e' la

seguente:

LINEA 10 Azzera lo schermo

LINEA 20 Imposta il puntatore dell'animazione al valore dal quale il COMMODORE 64 deve leggere i dati dell'animazione. L'animazione 0 e' posta a 2040, la 1 a 2041...la 7 a 2047. Si possono impostare tutti e 8 i puntatori delle animazioni a 13 sostituendo la linea 40 con:

FOR SP=2040TO2047:POKE SP,13:NEXT SP

LINEA 30 Inserisce la prima animazione (ANIMAZIONE 0) nei 63 bytes della memoria RAM del COMMODORE 64 a partire dalla posizione 832 (ogni animazione richiede 63 bytes). La prima animazione e' "indirizzata" nelle locazioni di memoria da 832 a 894.

LINEA 40 Uguaglia la variabile "V" a 53248, indirizzo di partenza del circuitovideo. Cio' permette di usare la forma (V + numero) per impostare l'animazione; inoltre, al momento dell'impostazione dell'animazione con l'istruzione POKE, tale forma non modifica la memoria, consentendo cosi' di lavorare con numeri piu' piccoli. Ad esempio, alla linea 50 abbiamo scritto POKE V+21: e' equivalente a POKE 53248+21 o a POKE 53269, solo che richiede meno spazio e si ricorda meglio.

LINEA 50 Abilita l'animazione 0. Ci sono 8 animazioni, numerate da 0 a 7. Per attivare un'animazione singola o un gruppo di esse, basta scrivere POKE V+21 seguito da un numero compreso fra 0 (disattiva tutte le animazioni) e 255 (attiva tutte le animazioni). La seguente tabella illustra come attivare e disattivare una o piu' animazioni:

TUTTO ON	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7	TUTTO OFF
V+21,255	V+21,1	V+21,2	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128	V+21,0

Si possono attivare anche combinazioni di animazioni: ad esmpio, POKE V+21,129 attiva le due animazioni 0 e 7 sommando i due numeri di attivazione (vd. tabella per la costruzione delle animazioni).

LINEA 60 Imposta il colore dell'animazione 0. Ci sono 16 colori a disposizione di un'animazione, numerati da 0 (nero) a 15 (grigio). Ogni animazione richiede una POKE diversa per ogni colore impostato da V+39 a V+46. POKE V+39 colora l'animazione 0 di bianco; POKE V+46,15 colora l'animazione 7 di grigio (vd. tabella per la costruzione delle animazioni).
Quando si crea un'animazione, questa rimane in memoria finche' non viene disattivata o ridefinita, oppure non viene spenta la macchina. Si puo' cosi' cambiare colore, disposizione e pefino luminosita' all'animazione, sia in modo DIRETTO che IMMEDIATO, quest'ultimo utile in caso di stampa. Ad esempio, far girare il precedente programma, poi battere in modo DIRETTO:

POKE V+39,8

è quindi RETURN : l'animazione sullo schermo e' diventata ARANCIONE; inserendo allo stesso modo i numeri da 0 a 15 si possono visualizzare

gli altri colori. Poiche' si e' agito in modo DIRETTO, rilanciando il programma l'animazione assume il colore originale (bianco).

LINEA 70 Determina la POSIZIONE ORIZZONTALE "X" dell'animazione sullo schermo. Questo numero rappresenta la locazione dell'ANGOLO IN ALTO A SINISTRA dell'animazione. La posizione orizzontale piu' lontana visibile a sinistra e' 24, sebbene si possa spostare l'animazione FUORI SCHERMO fino alla posizione 0.

LINEA 80 Determina la POSIZIONE VERTICALE Y dell'animazione sullo schermo. Il programma posiziona l'animazione nel punto (X=24, Y=100), rispettivamente posizione orizzontale e verticale. Un'altra posizione puo' essere ottenuta digitando:

```
POKE V,24:POKE V+1,50
```

Dopo aver battuto RETURN, la figura si sposta nell'angolo in alto a sinistra dello schermo; per portarla nell'angolo in basso a sinistra, digitare:

```
POKE V,24:POKE V+1,229
```

Ogni numero da 832 a 895 rappresenta, nell'indirizzo dell'animazione 0, un blocco di 8 pixel, con 3 blocchi di 8 pixel in ogni riga orizzontale dell'animazione. Il ciclo della linea 80 da' istruzione al computer per rendere pieni i primi, i secondi..., gli ultimi 8 pixel dell'angolo basso destro dell'animazione. Il funzionamento puo' essere osservato meglio digitando:

POKE 833,0 (per rimetterlo a posto POKE 833,255 o lanciare (RUN) il programma).

Come si puo' vedere, il secondo gruppo di 8 pixel e' stato cancellato; allo stesso modo, inserendo nel programma la seguente linea:

```
99 FORA=836TO891STEP3:POKEA,0:NEXT A
```

si cancellano i blocchi al centro dell'animazione. Va ricordato che i pixel di cui e' formata una figura sono raggruppati in blocchi di 8; la precendete linea cancella il quinto gruppo di 8 pixel (blocco 836) ed ogni terzo blocco fino al blocco 890 Introducendo tramite POKE uno qualunque degli altri numeri da 832 a 894, si puo' visualizzare ogni blocco da pieno (255) a vuoto (0).

COMPATTAZIONE DEI PROGRAMMI DI ANIMAZIONE

Qella illustrata di seguito e' un'utile tecnica di compattazione. Il programma illustrato in precedenza puo' essere ulteriormente abbreviato compattandolo opportunamente come illustrato di seguito:

```
10 PRINTCHR$(147):V=53248:POKEV+21,1:POKE2040,13:POKEV+39,1

20 FORS=832TO894:POKES,255:NEXT:POKEV,24:POKEV+1,100
```

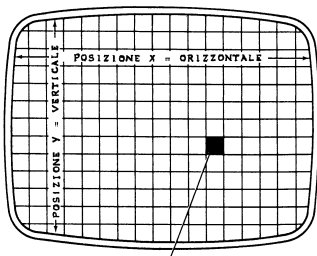
Altri modi di compattazione possono essere osservati nel paragrafo GUIDA ALLA COMPATTAZIONE.

Per essere visualizzata, un'animazione locata in questo punto deve avere impostata sia la posizione X (orizzontale) che la posizione Y (verticale)

Figura 3.4 - Divisione dello schermo video in una griglia di coordinate (X,Y)

POSIZIONAMENTO DELLE ANIMAZIONI SULLO SCHERMO

Tutto lo schermo video e' diviso in una griglia di coordinate X e Y, come una carta millimetrata. La COORDINATA X e' la posizione ORIZZONTALE sullo schermo, la Y quella VERTICALE (vd. figura 3.4).

Per posizionare qualunque animazione sullo schermo, bisogna usare POKE per due impostazioni - la posizione X e la posizione Y - che dicano al computer dove visualizzare L'ANGOLO IN ALTO A SINISTRA dell'animazione. Va ricordato che un'animazione e' formata da 504 punti, 24 orizzontali X 21 verticali; percio', quando si posiziona un'animazione nell'angolo in alto a sinistra dello schermo, l'animazione viene visualizzata come un'immagine grafica di 24 pixel ORIZZONTALI X 21 VERTICALI, iniziando dalla posizione (X,Y) precedentemente definita. L'animazione viene visualizzata partendo dall'angolo in alto a sinistra dell'intera animazione, anche se l'animazione viene definita usando solo una parte dell'area animazione di 24 X 21 pixel.

La seguente figura 3.5 illustra il funzionamento delle posizioni X e Y sullo schermo; l'area grigia indica il campo visibile sullo schermo, quella bianca la zona FUORI da tale campo.

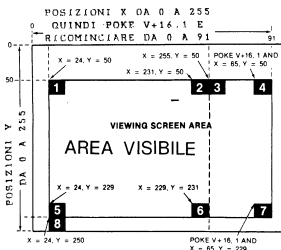



Figura 3.5 - Determinazione delle posizioni X-Y di un'animazione

Per visualizzare un'animazione in una data locazione, occorre impostare con una POKE i valori di X e Y per ogni animazione, tenendo presente che ogni animazione ha la propria POKE X e POKE Y. I valori di X e Y di tutte e 8 le animazioni sono i seguenti:

INSERIRE CON UNA POKE QUESTI VALORI PER DETERMINARE LE POSIZIONI X-Y DELL'ANIMAZIONE

	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7
X	V,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
X di DESTRA	V+16,1	V+16,2	V+16,4	V+16,8	V+16,16	V+16,32	V+16,64	V+16,128

IMPOSTAZIONE DELLA POSIZIONE X - Questi valori sono compresi fra 0 e 255, partendo da sinistra. I valori da 0 a 23 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE FINO ALLA 255-esima POSIZIONE (per le posizioni oltre la 256-esima, si veda il prossimo paragrafo). Per collocare l'animazione in una di queste posizioni, usare POKE con la corrispondente POSIZIONE X; ad esempio, POKE V+2,24 posiziona l'animazione 1 nella posizione X piu' a sinistra NEL CAMPO VISIBILE.

VALORI DI X OLTRE LA 255-ESIMA POSIZIONE - Per posizionare un'animazione oltre la 255-esima posizione, e' necessaria una seconda POKE che usa i valori illustrati nella riga "X DI DESTRA" della tabella 3.5. Di solito, la numerazione orizzontale continua oltre 255, ma poiche' i registri contengono solamente 8 bit, bisogna usare un secondo registro per accedere alla PARTE DESTRA dello schermo, ricominciando la numerazione da 0. Quindi, per oltrepassare la posizione 255, bisogna impostare POKE V+16, seguito dal numero dell'animazione. Si guadagnano cosi' altre 65 posizioni orizzontali

(rinumerate da 0 a 65) nella zona visibile a DESTRA dello schermo (impostando il valore delle X sulla destra a 255, si esce dal bordo destro dello schermo visibile)

IMPOSTAZIONE DELLA POSIZIONE Y - Questi valori sono compresi fra 0 e 255, a partire dall'alto. I valori compresi fra 0 e 49 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE NELLA PARTE ALTA dello schermo; quelli compresi fra 50 e 229 la collocano nel CAMPO VISIBILE; quelli compresi fra 230 e 255 collocano tutta l'animazione FUORI DAL CAMPO VISIBILE NELLA PARTE BASSA dello schermo.
Consideriamo il seguente programma:

```
              SHIFT CLR/HOME
10 PRINT"♥": V=53248:POKEV+21,2:POKE2041,13
15 FORS=832TO895:POKES,255:NEXT
20 POKEV+40,7
30 POKEV+2,24
40 POKEV+3,50
```

Questo semplice esempio illustra quali valori assegnare alle coordinate X e Y dell'animazione 1, un cubo pieno, nel caso che si voglia collocare tale animazione nell'angolo in alto a sinistra dello schermo. Se si modifica la linea 40 nel modo seguente:

```
40 POKE V+3,229
```

L'animazione viene visualizzata nell'angolo in basso a sinistra. Modificando invece la linea 30 in:

```
30 POKE V+2,255
```

l'animazione viene spostata sul LIMITE DESTRO DI X (255), per cui il bit piu' significativo del registro 16 deve essere IMPOSTATO. In altre parole, per RIPOSIZIONARE il contatore della posizione X al valore 256 (256-esimo pixel) dello schermo, occorre impostare POKE V+16 seguito dal numero mostrato nella colonna X di DESTRA della precedente tabella delle posizioni di X e Y. Quindi la linea 30 diventa:

```
30 POKE V+16,PEEK(V+16)OR 2:POKE V+2,0
```

POKE V+16,2 imposta il bit piu' significativo della posizione X dell'animazione 1, riposizionando a 256 (256-esimo pixel dello schermo). POKE V+2,0 visualizza l'animazione nella NUOVA POSIZIONE ZERO, che attualmente e' il 256-esimo pixel.
Per riposizionarsi sulla sinistra dello schermo, occorre impostare daccapo a 0 il bit piu' significativo del contatore della posizione X scrivendo:

```
POKE V+16, PEEK(V+16)AND 253
```

RIASSUMENDO: la POSIZIONE X di ogni animazione assume tutti i valori compresi fra 0 e 255. Per accedere alle posizioni dello schermo oltre a 255 (255-esimo pixel), si deve usare una POKE V+16 aggiuntiva, che imposti il bit piu' significativo della posizione X e ricominci a contare da 0 al 256-esimo pixel in poi (ad esempio, POKE V+16, PEEK(V+16)OR1 e POKE V,1 devono essere usate per posizionare l'animazione 0 al 257-esimo pixel dello schermo). Per tornare nella

posizione X di sinistra, bisogna DISATTIVARE l'impostazione del controllo digitando POKE V+16,PEEK(V+16)AND 254.

POSIZIONAMENTO SULLO SCHERMO DI PIÙ ANIMAZIONI

Il seguente programma definisce TRE DIFFERENTI ANIMAZIONI (0,1 e 2) di colore diverso, posizionandole in tre diversi punti dello schermo:

```
10 PRINT"[SHIFT CLR HOME]":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKE+41,8
50 POKEV,24:POKEV+1,50
60 POKEV+2,12:POKEV+3,229
70 POKEV+4,255:POKEV+5,50
```

Per comodita', le tre animazioni sono state definite come quadrati pieni che traggono i dati dalla stessa fonte: l'aspetto che qui interessa e' il loro posizionamento sullo schermo. L'animazione 0 (bianca) si trova nell'angolo in alto a sinistra; l'animazione 1 (gialla) in basso a sinistra PER META' FUORI SCHERMO (ricordiamo ancora una volta che 24 e' la posizione piu' a sinistra del campo visibile, per cui una posizione di X minore di 24 pone una parte o tutta l'animazione fuori dallo schermo; per questo si e' usata per X la posizione 12, che visualizza meta' animazione lasciando l'altra meta' fuori dallo schermo); infine, l'animazione 2 (arancione) si trova nella POSIZIONE LIMITE DI DESTRA (posizione 255). Che cosa bisognerebbe fare se si volesse visualizzare un'animazione nella zona di DESTRA DELLA POSIZIONE 255 di X? Lo saprete...al prossimo paragrafo!

VISUALIZZAZIONE DI UN'ANIMAZIONE OLTRE LA 255-ESIMA POSIZIONE DI X

Questa visualizzazione richiede una particolare POKE che imposti il bit piu' significativo della posizione X ed inizi alla 256-esima posizione (256-esimo pixel).
Innanzi tutto, occorre impostare POKE V+16 seguito dal numero dell'animazione che si vuole usare (supponiamo di usare l'animazione 0, i cui valori di X e Y devono essere rintracciati nella TAVOLA DELLE POSIZIONI X e Y). All'assegnazione della posizione X, occorre tenere presente che il contatore di X varia da 0 a 256; la linea 50 diventa quindi:

```
50 POKE V+16,1:POKE V,24:POKE V+1,75
```

La POKE V+16 di questa linea consente di "aprire" la parte detstra dello schermo del numero di posizioni richieste. Ora la nuova posizione 24 di X inizia, per l'animazione 0, a DESTRA dello schermo; modificare la linea 60 come segue:

```
60 POKE V+16,1:POKE V,65:POKE V+1,75
```

Alcune prove fatte con i valori riportati nella tabella delle posizioni X e Y forniscono i valori necessari per posizionare e muovere le animazioni da destra a sinistra. Il posizionamento delle animazioni viene ulteriormente illustrato nel paragrafo riguardante il movimento delle animazioni.

PRIORITÀ DELLE ANIMAZIONI

Precedentemente si e' esposto il principio in base al quale le animazioni sembrano muoversi DAVANTI o DIETRO l'una rispetto all'altra. Quest'illusione tridimensionale si ottiene per mezzo delle PRIORITA' DELLE ANIMAZIONI del sistema, che determinano quali animazioni hanno priorita' maggiore rispetto alle altre nel caso in cui due o piu' animazioni SI SOVRAPPONGANO sullo schermo.
La regola fondamentale e' "primo entrato, primo servito" (FIFO=First In First Out), cioe' le animazioni di bassa numerazione hanno AUTOMATICAMENTE priorita' maggiore di quelle di alta numerazione. Se, ad esempio, si visualizzano le animazioni 0 e 1 in modo che si sovrappongano, l'animazione 0 appare DAVANTI alla 1. Normalmente, l'animazione 0 ha la priorita' PIU' ALTA, essendo 0 il numero piu' basso a disposizione delle animazioni. Analogamente, l'animazione 1 ha la priorita' maggiore delle 2-7, la 2 delle 3-7, ecc.; infine, l'animazione 7 ha la priorita' PIU' BASSA DI TUTTE, ed appare sempre "DIETRO" ALLE ALTRE ANIMAZIONI CHE LE SI SOVRAPPONGONO.
Per poter vedere come funzionano le priorita', cambiare le linee 50, 60 e 70 del precedente programma nel modo seguente:

```
         SHIFT CLR HOME
10 PRINT"♥":V=53248:FORS=832TO895:POKES,255:NEXT
20 FORM=2040TO2042:POKEM,13:NEXT
30 POKEV+21,7
40 POKEV+39,1:POKEV+40,7:POKEV+41,8
50 POKEV,24:POKEV+1,50:POKEV+16,0
60 POKEV+2,34:POKEV+3,60
70 POKEV+4,44:POKEV5,70
```

Dovrebbe comparire un'animazione bianca sopra un'animazione gialla sopra un'animazione arancione. Ovviamente, si puo' trarre vantaggio dalle priorita' per MUOVERE LE ANIMAZIONI e migliorare i programmi che le riguardano.

COME DISEGNARE UN'ANIMAZIONE

Disegnare un'animazione COMMODORE e' come colorare degli spazi vuoti di un libro da disegni. Ogni animazione e' formata da un insieme di punti chiamati pixel; il disegno di un'animazione viene quindi ricondotto a "colorare" alcuni pixel.
La griglia della figura 3.6 e' simile ad un'animazione vuota.

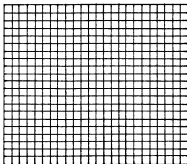



Figura 3.6 - Griglia per la costruzione di animazioni

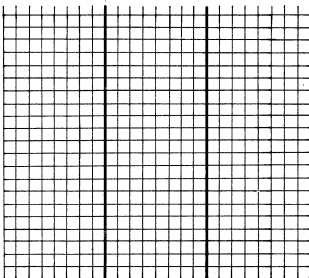
Ogni quadratino rappresenta un pixel dell'animazione; la griglia e' formata da 24 pixel X 21, pari a 504 pixel dell'animazione completa. Per costruire un'animazione che assomigli a qualcosa, occorre colorare questi pixel usando uno speciale PROGRAMMA...m come si riesce a controllare piu' di 500 punti indipendenti? Ecco venire in aiuto la programmazione, che consente di trattare, per ogni animazione, 63 numeri invece dei 504 richiesti.

CREAZIONE DI UN'ANIMAZIONE...MINUTO PER MINUTO

La creazione di un'animazione puo' essere riassunta nei seguenti passi:

PASSO 1:

Scrivere SU UN FOGLIO DI CARTA il programma di creazione di un'animazione illustrato sotto. Da notare la linea 100, che da' il via ad una speciale sezione di istruzioni DATA contenenti i 63 numeri necessari a creare un'animazione.

```
10 PRINT "♥":POKE53280,5:POKE53281,6      (SHIFT CLR HOME)
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
```

PASSO 2:

Colorare i pixel (i quadratini) della griglia della figura 3.6 (usare in alternativa un foglio di carta millimetrata, ricordando sempre che un'animazione e' formata da 24 pixel orizzontali X 21 verticali). si puo' disegnare qualunque immagine si desideri, ma per il nostro esempio disegneremo una scatola.

PASSO 3:

Consideriamo i primi 8 pixel: ogni colonna di pixel ha un numero, corrispondente ad una potenza del 2, da 128 (=2 7) a 1 (=2 0). La particolare addizione che stiamo per fare e' un tipo di ARITMETICA BINARIA molto usata dai calcolatori come modo particolare di conteggio. Gli 8 pixel considerati sono visualizzati in dettaglio qui di seguito.

128	64	32	16	8	4	2	1

PASSO 4:

Sommare i numeri dei pixel PIENI. Poiche' il primo gruppo di pixel e' pieno, il loro totale e' 255

PASSO 5:

Introdurre tale totale come PRIMO ELEMENTO DELL'ISTRUZIONE DATA della linea 100 del programma di costruzione di un'animazione dato piu' sotto.

PASSO 6:

Consideriamo I PRIMI 8 PIXEL DELLA SECONDA RIGA dell'animazione. Sommare anche in questo caso i valori dei pixel pieni. Poiche' solo uno di questi 8 pixel e' pieno, il totale e' 128. Introdurre questo totale come primo elemento dell'istruzione DATA nella linea 101

128	64	32	16	8	4	2	1

PASSO 7:

Sommare i valori del gruppo successivo di 8 pixel (0 in quanto sono tutti VUOTI) e posizionarli in ordine nella linea 101. Passare quindi al gruppo seguente di pixel e ripetere l'operazione per TUTTI I GRUPPI DI 8 PIXEL (3 gruppi per ogni riga, per 21 righe). Alla fine si ha un totale di 63 numeri, ognuno dei quali rappresenta un gruppo di 8 pixel, che moltiplicato per 63 da' un totale di 504 punti indipendenti. Il programma puo' essere visto anche nel modo seguente. Ogni riga del programma rappresenta UNA RIGA dell'animazione. Un gruppo si tre numeri di ogni riga rappresenta UN GRUPPO DI 8 PIXEL. Ogni numero comunica al computer quali pixel riempire e quali lasciare vuoti.

PASSO 8:

COMPATTARE IL PROGRAMMA IN MENO SPAZIO RAGGRUPPANDO TUTTE LE ISTRUZIONI DATA, COME ILLUSTRATO NEL PROGRAMMA SEGUENTE. Appare ora chiaro il motivo per cui si e' consigliato di scrivere il programma dell'animazione su un foglio di carta: LE ISTRUZIONI DATA DELLE LINEE

DA 100 A 120 del programma di cui al PASSO 1 sono state scritte in quel modo solamente per aiutare a visualizzare quali numeri mettere in relazione ai gruppi di pixel dell'animazione. Il programma finale puo' dunque essere "compattato" come segue:

```
10 PRINT" ":POKE53280,5:POKE53281,6
20 V=53248:POKEV+34,3
30 POKE53269,4:POKE2042,13
40 FORN=0TO62:READQ:POKE832+N,Q:NEXT
100 DATA255,255,255,128,0,1,128,0,1,128,0,1,144,0,1,144,0,1,144,0,
1,144,0,1
101 DATA144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,144,0,1,128,0,1
128,0,1
102 DATA128,0,1,128,0,1,128,0,1,128,0,1,255,255,255
200 X=200:Y=100:POKE53252,X:POKE53253,Y
```

MOVIMENTO DELLE ANIMAZIONI SULLO SCHERMO

Per muovere lentamente l'animazione sullo schermo, aggiungere al programma precedente le seguenti due linee:

```
50 POKEV+5,100:FORX=24TO255:POKE V+4,X:NEXT:POKEV+16,4
55 FORX=0TO65:POKEV+4,X:NEXTX:POKEV+16,0:GOTO50
```

LINEA 50 Imposta la posizione Y a 100(provare, tanto per cambiare, i valori 50 e 229). Il successivo LOOP FOR...NEXT muove l'animazione dalla posizione 0 alla 255, rispettivamente. Quando quest'ultima viene raggiunta,viene impostata la posizione X di destra (POKE V+16,2) necessaria per attraversare la parte destra dello schermo.

LINEA 55 Contiene un loop For...Next che muove l'animazione nelle successive 65 righe dello schermo.Da notare che il valore di X e' stato rimesso a zero, ma poiche' si e' impostato l'X DI DESTRA (POKE V+16,2), X parte dalla destra dello schermo.

Queste due linee sono in ciclo infinito (GOTO 50); se si desidera che l'animazione attraversi una sola volta lo schermo e scompaia, togliere GOTO 50

Le seguenti istruzioni muovono l'animazione AVANTI e INDIETRO:

```
50 POKE V+5,100:FOR X=24TO255:POKE V++4,X:NEXT:POKEV+16,4
51 FOR X=0TO65: POKE V+4,X:NEXT X
55 FOR X=65TO0 STEP-1:POKE V+4,X:NEXT:POKE V+16,0
56 FOR X=255TO24 STEP-1:POKE V+4,X:NEXT
60 GOTO 50
```

Questa versione del programma e' analoga alla precedente, solo che quando l'animazione raggiunge il bordo destro dello schermo, GIRA SU SE' STESSA e torna indietro. Cio'e' ottenuto per mezzo di STEP-1, che dice al programma di posizionare (POKE) l'animazione nei valori di destra compresi fra 65 e 0, poi in quelli di sinistra da 255 a 0 tornando indietro di una posizione alla volta.

SCROLLING VERTICALE

Questo tipo di movimento dell'animazione e' chiamato scrolling; per la sua realizzazione su e giu' alla posizione Y, basta UNA SOLA LINEA. CANCELLARE LE PRECEDENTI LINEE 50 e 55 battendo i rispettivi numeri di linea seguiti da RETURN:

```
50 ( RETURN )
55 ( RETURN )
```

e quindi impostare la LINEA 50 come segue:

```
50 POKE V+4,24:FOR Y=0TO255:POKE V+5,Y:NEXT
```

IL TOPOLINO BALLERINO – UN ESEMPIO DI PROGRAMMA DI ANIMAZIONE

Le tecniche descritte in una guida per programmatori sono talvolta di difficile comprensione, ragion per cui presentiamo qui di seguito un divertente programma di animazione chiamato "TOPOLINO BALLERINO". Questo programma usa tre differenti animazioni in rapido movimento unite ad effetti sonori. Quella seguente e' la descrizione di OGNI ISTRUZIONE, che riteniamo possa aiutare a comprendere la costruzione ed il funzionamento del programma.

```
5 S=54272:POKES+24,15:POKES,220:POKES+1,68:POKES+5,15:POKES+6,215
10 POKES+7,120:POKES+8,100:POKES+12,15:POKES+13,215
          SHIFT CLR HOME
15 PRINT"♥" V=53248:POKEV+21,1
20 FORS1=12288TO12350:READQ1:POKES1,Q1:NEXT
25 FORS2=12352TO12414:READQ2:POKES2,Q2:NEXT
20 FORS3=12416TO12478:READQ3:POKES3,Q3:NEXT
35 POKEV+39,15:POKEV+1,68
                  CTRL 2                  C= 7
40 PRINTTAB(160) " I AM THE DANCING MOUSE!♥"
45 P=192
50 FORX=0TO347STEP3
55 RX=INT(X/256):LX=X-RX*256
60 POKEV,LX:POKEV+16,RX
70 IFP=192THENGOSUB200
75 IFP=193THENGOSUB300
80 POKE2040,P:FORT=1TO60:NEXT
85 P=P+1:IFP>194THENF=192
90 NEXT
95 END
100 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
101 DATA63,255,251,31,187,248,3,187,192,1,255,128,3,189,192,1,231,
    128,1,255,0
102 DATA31,255,0,0,124,1,1,254,0,1,199,32,3,131,224,7,1,192,1,192,0,
    3,192,0
103 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
104 DATA63,255,252,31,221,248,3,221,192,1,255,128,3,255,192,1,195,
    128,1,231,3
105 DATA31,255,255,0,124,0,0,254,0,1,199,0,7,1,128,7,0,204,1,128,
    124,7,128,56
106 DATA30,0,120,63,0,252,127,129,254,127,129,254,127,189,254,127,
    255,254
107 DATA63,255,252,31,221,248,3,221,192,1,255,134,3,189,204,1,199,
    152,1,255,48
108 DATA1,255,224,1,252,0,3,254,0
109 DATA7,14,0,204,14,0,248,56,0,112,112,0,0,60,0,-1
200 POKES+4,129:POKES+4,128:RETURN
300 POKES+11,129:POKES+11,128:RETURN
```

LINEA 5:

S=54272 — Uguaglia la variabile S a 54272, che e' la locazione di memoria di partenza del CIRCUITO SONORO. In seguito, anziche' impostare direttamente una locazione di memoria, usare una POKE S piu' un certo valore.

POKES+24,15 — Imposta il volume al massimo.

POKES,220 — Imposta una Bassa Frequenza nella Voce 1 per una nota che approssima un DO acuto in sesta ottava.

POKES+1,68 — Imposta un'Alta Frequenza nella Voce 1 per una nota che approssima un DO acuto in sesta ottava

POKES+5,15 — Imposta ATTACCARE/DECADERE per la VOCE 1; consiste in questo caso nel massimo livello di DECADERE senza alcun ATTACCARE, producendo cosi' un effetto "ECO".

POKES+6,215 — Imposta SUSTENERE/RILASCIARE per la Voce 1 (215 e' una combinazione di questi due valori).

LINEA 10:

POKES+7,120 — Imposta l'Alta Frequenza per la Voce 2

POKES+8,100 — Imposta la Bassa Frequenza per la Voce 2

POKE S+12,15 — Imposta ATTACK/DECAY per la Voce 2 allo stesso livello della precedente Voce 1

POKES+13,215 — Imposta ATTACCARE/DECADERE per la Voce 2 allo stesso livello della Voce 1

LINEA 15:

PRINT" SHIFT CLR/HOME " — Azzera lo schermo all'inizio del programma.

V=53248 — Definisce la variabile "V" come la locazione di partenza del circuito VIC-II che controlla le animazioni; in seguito, le locazioni dell'animazione saranno definite come V + un certo valore.

POKEV+21,1 — Abilita l'animazione 1

FORS1=12288 TO12350 — Attualmete si usa una sola animazione (la #0), ma che fa capo a tre insiemi di dati dell'animazione necessari alla formazione di tre diverse figure. Per avere l'animazione, occorre assegnare i puntatori dell'animazione 0 a tre diverse locazioni di memoria in cui si sono registrati i dati che definiscono le tre differenti figure. La stessa animazione viene via via ridefinita rapidamente come tre differenti figure per riprodurre l'animazione del Topolino Ballerino. Nelle istruzioni DATa, si possono definire quante figure si

desiderano e ruotarle attorno ad una o piu' animazioni; si puo' vedere, quindi, che non occorre limitare un'animazione ad una figura, o viceversa. Un'animazione puo'assumere molte figure differenti, semplicemente cambiando il puntatore che imposta quell'animazione nelle diverse locazioni di memoria dove sono registrati i dati dell'animazione relativi alle diverse figure. Questa linea significa che i dati per la figura 1 dell'animazione sono stati collocati nelle locazioni da 12288 a 12350.

READ Q1	Legge nell'ordine 63 numeri dalle istruzioni DATA che iniziano alla linea 100 Q1 e' un nome di variabile arbitrario
POKE S1,Q1	Posiziona il primo numero rilevato dalle istruzioni DATA (il primo, "Q1", e' 30) nella prima locazione di memoria che e' la 12288 Questa istruzione e' equivalente a POKE1288,30.
NEXT	Comunica al computer di considerare solo le parti comprese tra FOR e NEXT e di eseguire le istruzioni che incontra. In altre parole, l'istruzione NEXT fa leggere (READ) al computer il prossimo (NEXT) Q1 dalle istruzioni DATA. Takle Q1 e' 0, il che fa incrementare S1 di 1 unita' verso il valore successivo, che e'12289. L'istruzione NEXT fa eseguire il ciclo fino all'ultimo valore della serie, dato da POKE 12350,0.

LINEA 25:

FORS2=12352	La seconda figura dell'animazione 0 e' definita dai dati da 12352 a 12414 DA NOTARE che si e' saltata la locazione 12351: questa infatti, e' la 64-esima locazione usata nella definizione del primo gruppo dell'animazione. Occorre ricordare, quando si definiscono animazioni in locazioni consecutive, che si usano 64 locazioni, ma i dati delle animazioni si trovano solamente nelle prime 63.
READ Q2	Legge i 63 numeri che seguono quelli usati per la prima figura dell'animazione Questa READ cerca semplicemente il numero immediatamente successivo nella zona occupata da DATA e comincia a leggere 63 numeri, uno alla volta.
POKES2,A2	Posiziona i dati (Q2) nelle locazioni di memoria (S2) della seconda figura dell'animazione, che inizia alla locazione 12352.
NEXT	Come la precedente linea 20

LINEA 30:

FORS3=12416 TO12478	La terza figura dell'animazione 0 e' definita dalle istruzioni DATA locate da 12416 a 12478.
READQ3	Legge in Q3 gli ultimi 63 numeri in ordine.
POKE S3,A3	Posiziona tali numeri nelle locazioni da 12416 a 12478.
NEXT	Analogo alle linee 20 e 25.

LINEA 35:

POKE V+39,15	Imposta il colore grigio chiaro per l'animazione 0.
POKE V+1,68	Imposta l'angolo in alto a destra del quadrato dell'animazione nella posizione verticale (Y) 68. Analogamente la posizione 50 e' la posizione Y dell'angolo in alto a sinistra dello schermo.

LINEA 40:

PRINT TAB(160)	Tabula 160 spazi dello spazio carattere in alto a sinistra, cioe' si posiziona 4 righe piu' in basso della istruzione di azzeramento dello schermo; in altre parole, il messaggio da visualizzare inizia alla sesta riga dello schermo.
"CTRL WHT	Premendo contemporaneamente questi due tasti dopo le virgolette, viene visualizzata una E "reverse": il colore di qualunque cosa visualizzata da ora in poi viene impostato a bianco.
J AM THE DANCING MOUSE!	Messaggio visualizzato dalla PRINT
C 7"	Ripristina il colore blu dopo l'istruzione PRINT. Premendo questi due tasti dopo le virgolette si visualizza un asterisco (*) "reverse".

LINEA 45:

P=192	Uguaglia la variabile P a 192. Questo numero e' il puntatore da usare per "puntare", in questo caso, l'animazione 0 alla locazione di memoria che comincia a 12288. Il segreto per usare un'animazione allo scopo di creare un effetto di movimento composto da tre diverse figure sta nello spostare questo puntatore alle locazioni delle altre due figure dell'animazione.

LINEA 50:

FORX=0TO347 STEP3	Incrementa il movimento dell'animazione di 3X posizioni alla volta, dalla posizione 0 alla 347.

LINEA 55:

RX = INT(X/256)	Parte intera di X/256: indica che RX assume il valore 0 per X < 256, a quando X = 256. Si usa RX al momento di attivare la PARTE DESTRA dello schermo, impostandò POKE V+16 a 0 o a 1
LX=X-RX*256	Quando l'animazione e' nella posizione 0 di X, questa formula diviene LX=0-0*256=0; quando invece si trova nella posizione 1 di X, la formula diviene LX=1-0*256=1; quando infine si trova nella posizione 256 di X, si ottiene LX=256-1*256=0, mediante il quale si imposta X di nuovo a 0, operazione effettuata quando ci si muove sulla DESTRA dello schermo (POKE V+16,1).

LINEA 60:

POKEV,LX	Imposta POKE V, senza alcun inremento, insieme ad un altro valore, quando si desidera impostare sullo schermo la POSIZIONE ORIZZONTALE (X) dell'animazione (vd. tabella per la creazione di un'animazione). Come mostrato in precedenza, il valore della posizione orizzontale LX dell'animazione varia da 0 a 255; quando quest'ultima raggiunge questo valore, viene automaticamente riportata a 0 dall'espressione di LX impostata alla linea 55
POKE V+16,RX	Questa istruzione abilita sempre la DESTRA dello schermo oltre la posizione 256, e riporta a zero le coordinate del posizionamento orizzontale. In base alla posizione dell'animazione, determinata dalla espressione di RX alla LINEA 55, RX puo' essere 0 oppure 1.

LINEA 70:

IFP=192THEN GOSUB200	Se il puntatore all'animazione e' impostato a 192 (prima figura dell'animazione), il controllo della forma d'onda per il primo effetto sonoro viene impostato, alla linea 200, a 129 e 128

LINEA 75:

IFP=193THEN GOSUB300	Se il puntatore all'animazione e' impostato a 193 (seconda figura dell'animazione), il controllo della forma d'onda per il secondo effetto sonoro (Voce 2) viene impostato, alla linea 300, a 129 e 128

LINEA 80:

POKE2040,P	Imposta il puntatore dell' animazione alla locazione 192 (come si ricordera', P e' stato posto uguale a 192 alla linea 45).
FORT=1TO60: NEXT	Loop di ritardo per impostare il ritmo della danza del topolino (tale ritmo puo' essere variato aumentando o diminuendo il numero 60).

LINEA 85:

P=P+1 Aggiunge 1 al valore iniziale del puntatore.

IFP>194 THENP=192 Punta l'animazione a 3 sole locazioni di memoria. Il valore 192 punta alle locazioni da 12288 a 12350; 193 punta alle locazioni da 12416 a 12478. Quesra linea imposta daccapo P a 192 non appena P diventa 195, impedendo cosi' a P stesso di assumere quest'ultimo valore. In questo modo il Puntatore spazia consecutivamente le tre figure dell'animazione all'interno dei gruppi di 64 byte delle locazioni di memoria contenenti i dati.

LINEA 90:

NEXT X Solamente dopo che l'animazione ha assunto una delle tre figure definite dalle istruzioni DATA, l'animazione puo' muoversi sullo schermo. A questo punto l'animazione si muove di 3X posizioni alla volta (anziche' spostarsi lentamente di una, come pure e' possibile). Saltando 3 posizioni alla volta (istruzione STEP), il Topilino "danza"
piu' velocemente. NEXT X chiude il LOOP, aperto alla linea 50, che determina la posizione X.

LINEA 95:

END Chiude il programma, quando l'animazione esce dallo schermo.

LINEA 100-109:

DATA Le figure dell'animazione vengono lette in ordine dai numeri delle istruzioni DATA. Prima i 63 numeri che compongono la Figura 1, poi i 63 della Figura 2, infine i 63 della figura 2, quindi i 63 della figura 3. Questi dati vengono letti permanentemente nelle 3 locazioni di memoria, dopodiche' il programma punta l'animazione 0 alle 3 locazioni di memoria e l'animazione assume automaticamente la figura rappresentata dai dati presenti in quelle locazioni. Puntando l'animazione ad una figura alla volta si ottiene l'effetto "movimento". Per osservare come questi numeri influenzano ogni animazione, basta sostituire i primi 3 numeri della linea 100 con 255,255,255. Si veda anche il paragrafo sulla definizione delle figure di un'animazione.

LINEA 200:

POKES+4,129 Attiva l'effetto sonoro.

POKES+4,128 Disattiva l'effetto sonoro.

RETURN Riporta il programma alla fine della LINEA 70 dopo aver cambiato impostazione del controllo della forma d'onda; l'elaborazione riprende dalla fine della LINEA 70.

LINEA 300:

POKES+11,129 Attiva l'effetto sonoro.

POKES+11,128 Disattiva l'effetto sonoro.

RETURN Riporta l'elaborazione alla fine della LINEA 75.

PRONTUARIO PER LA COSTRUZIONE DI UN'ANIMAZIONE

	ANIM. 0	ANIM. 1	ANIM. 2	ANIM. 3	ANIM. 4	ANIM. 5	ANIM. 6	ANIM. 7
Attiva una animazione	V+21,2	V+21,1	V+21,4	V+21,8	V+21,16	V+21,32	V+21,64	V+21,128
Caricamento in memoria (imposta i puntatori)	2040, 192	2041, 193	2042, 194	2043, 195	2044, 196	2045, 197	2046, 198	2047, 199
Locazione dei pixel (da 12288 a 12798)	12288 to 12350	12352 to 12414	12416 to 12478	12480 to 12542	12544 to 12606	12608 to 12670	12672 to 12734	12736 to 12798
Colore animazione	V+39,C	V+40,C	V+41,C	V+42,C	V+43,C	V+44,C	V+45,C	V+46,C
Posizione X di sinistra (0-255)	V+0,X	V+2,X	V+4,X	V+6,X	V+8,X	V+10,X	V+12,X	V+14,X
Posizione X di destra (0-255)	V+16,1 V+0,X	V+16,2 V+2,X	V+16,4 V+4,X	V+16,8 V+6,X	V+16,16 V+8,X	V+16,32 V+10,X	V+16,64 V+12,X	V+16,128 V+14,X
Posizione Y	V+1,Y	V+3,Y	V+5,Y	V+7,Y	V+9,Y	V+11,Y	V+13,Y	V+15,Y
Espansione orizzontale	V+29,1	V+29,2	V+29,4	V+29,8	V+29,16	V+29,32	V+29,64	V+29,128
Espansione verticale	V+23,1	V+23,2	V+23,4	V+23,8	V+23,16	V+23,32	V+23,64	V+23,128
Attivazione modo multicolore	V+28,1	V+28,2	V+28,4	V+28,8	V+28,16	V+28,32	V+28,64	V+28,128
Multicolore 1 (primo colore)	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C	V+37,C
Multicolore 2 (secondo colore)	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C	V+38,C
Priorità animazioni	Animazioni di numero piu' basso hanno priorita' di schermo maggiore delle animazioni di numero piu' alto. Ad esempio, l'animazione 0 ha priorita' su TUTTE le altre animazioni, e l'animazione 7 ha la priorita' piu' bassa. Cio' significa che le animazioni di numero piu' basso sembrano sempre muoversi DAVANTI o SOPRA le animazioni di numero piu' alto							
Contatto (animazione-animazione)	V+30 IF PEEK(V+30)ANDX=X THEN [azione]							
Contatto (animazione-fondo)	V+31 IF PEEK(V+31)ANDX=X THEN [azione]							

NOTE PER LA COSTRUZIONE DI UN'ANIMAZIONE

LOCAZIONI DI MEMORIA E PUNTATORI ALLA MEMORIA ANIMAZIONE ALTERNATIVI PER L'USO DEL BUFFER DEL REGISTRATORE

Memorizzazione (imposta i puntatori)	ANIM. 0	ANIM. 1	ANIM. 2	Se si pensa di usare da 1 a 3 animazioni, per il buffer del registratore si possono usare le locazioni 832-1023, ma per piu' di 3 animazioni si consigliano le locazioni 12288-12798 (vd. tabella).
	2040,13	2041,14	2045,15	
Pixel animaz. Locazioni per BlocchI 13-15	832 fino a 894	896 fino a 958	960 fino a 1022	

ATTIVAZIONE DI UN'ANIMAZIONE

Si puo' attivare una singola animazione usando una POKE V+21 e il numero estratto dalla tabella, ma l'attivazionre di UNA SOLA animazione DISATTIVA le altre. Per attivare DUE o PIU' animazioni, SOMMARE i numeri delle animazioni che si vuole attivare (es. POKE V+21,6 attiva le animazioni 1 e 2). Il seguente esmpio indica come attivare e disattivare un'animazione senza influenzare le altre.

ESEMPIO:

Per disattivare la sola animazione 0: POKEV+21,PEEKV+21AND(255-1). Cambiare il numero 1 di (255-1) in 1, 2, 4, 8, 16, 32, 64, 128 per le animazioni da 0 a 7. Per riattivare l'animazione senza influenzare le altre gia' attivate, usare POKEV+21,PEEK(V+21)OR1 cambiando OR1 in OR2 (ANIMAZIONE 2),OR3 (ANIMAZIONE 3), ecc.

VALORI DELLA POSIZIONE X OLTRE 255

Le posizioni di X vanno da 0 a 255 e poi RIPARTONO da 0 a 255. Per posizionare un'animazione oltre la posizione 255 di X sulla destra dello schermo, impostare prima POKE V+16 come gia' indicato, poi usare POKE per impostare un nuovo valore di X da 0 a 63, che posiziona l'animazione in una delle posizioni di X a destra dello schermo. Per ritornare alle posizioni 0-255, usare POKE V+16,0, quindi una seconda POKE i cui valori di X siano compresi fra 0 e 255.

VALORI DELLA POSIZIONE Y

Queste posizioni vanno da 0 a 255, comprendenti i valori da 0 a 49 fuori dalla parte superiore del quadro, quelli da 50 a 229 nel quadro, e quelli da 230 a 255 fuori dalla parte inferiore del quadro.

COLORI DI UN'ANIMAZIONE

Per colorare di bianco l'animazione 0, battere POKE V+39,1 (usare le IMPOSTAZIONI DEL COLORE TRAMITE POKE illustrate in tabella, ed i codici di colore individuale illustrati sotto):

0-NERO	4-PORPORA	8-ARANCIO	12-GRIGIO MEDIO
1-BIANCO	5-VERDE	9-MARRONE	13-VERDE CHIARO
2-ROSSO	6-BLU	10-ROSSO CHIARO	14-BLU CHIARO
3-AZZURRO	7-GIALLO	11-GRIGIO SCURO	15-GRIGIO CHIARO

LOCAZIONE DELLA MEMORIA

Per ogni animazione si deve riservare, nella memoria del computer, un BLOCCO separato di 64 BYTE, dei quali 63 vengono usati dai dati dell'animazione. Le impostazioni della memoria che seguono sono consigliate per l'impostazione del puntatore dell'animazione della tavola precedente. Ogni animazione e' unica e puo' essere definita come si desidera. Per fare tutte le animazioni uguali, puntare le animazioni che si desiderano uguali allo stesso registro delle animazioni.

DIFFERENTI IMPOSTAZIONI DEL PUNTATORE DELL'ANIMAZIONE

Questi sono SOLAMENTE ACCORGIMENTI da seguire per impostare il puntatore di un'animazione.
ATTENZIONE - I puntatori dell'animazione possono essere piazzati dovunque nella RAM, ma se sono troppo in "basso" un LUNGO PROGRAMMA BASIC si puo' sovrapporre ai dati dell'animazione, o viceversa. Per evitare cio', si possono piazzare le animazioni in un'area di memoria piu' alta (ad esempio 2040,192 per l'animazione 0 alle locazioni da 12288 a 12350, 2041,193 per l'animazione 1 alle locazioni da 12352 a 12414, ecc.); adeguando le locazioni di memoria da cui le animazioni traggono i "dati", si possono definire la bellezza di 64 differenti animazioni oltre a un notevole programma BASIC. A questo fine, occorre definire diverse "figure" di un'animazione nelle istruzioni DATA e poi ridefinire una particolare animazione modificando il "puntatore" in modo che tale animazione venga "puntata" a differenti aree di memoria contenenti differenti dati per la rappresentazione dell'animazione. Un esempio di questo funzionamento e' dato nel "TOPOLINIO BALLERINO" al quale si rimanda. Se si vuole che due o piu' animazioni assumano la stessa figura (pur cambiando posizione e colore di ciascuna), usare lo stesso puntatore e la stessa locazione di memoria delle animazioni che si vogliono uguagliare (ad esempio si possono puntare alla stessa locazione le animazioni 0 e 1 usando POKE 2040,192 e POKE 2041,192).

PRIORITÀ

Significa che un'animazione viene visualizzata davanti o dietro ad un'altra. Le animazioni di maggiore priorita' compaiono sempre davanti o sopra quelle di priorita' minore. La regola e' che animazioni di bassa numerazione hanno la priorita' su quelle di alta numerazione.
L'animazione 0 ha priorita' massima, la 7 minima; l'animazione 1 ha priorita' su 2-7, ecc. Mettendo due animazioni nella stessa posizione, quella di priorita' maggiore appare davanti a quella di priorita' minore. L'animazione di priorita' minore viene oscurata oppure "mostrata attraverso" quella di priorita' maggiore.

USO DEL MULTICOLORE

Si possono creare animazioni multicolore, anche se questa tecnica richiede l'uso di coppie di pixel (in altre parole, ogni "punto" o "blocco" colorato dell'animazione e' formato dal doppio di pixel per ogni lato). Si hanno a disposizione 4 colori: colore animazione (tabella precedente), multicolor 1, multicolor 2 e "colore di fondo" (ottenuto usando impostazioni a 0 che lasciano "intravedere il colore di fondo). Consideriamo un blocco di 8 pixel orizzontali del disegno di un'animazione. Il colore di ogni coppia di pixel e' determinato a

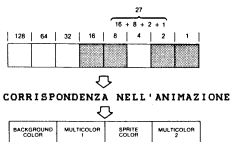
seconda che sia pieno il pixel di destra sinistra, entrambi o nessuno:

SOTTOFONDO (Se ENTRAMBI I PIXEL SONO VUOTI (zero), appare il colore di fondo dello schermo)

MULTICOLORE 1 (Se il PIXEL DI DESTRA di una coppia di pixel e' PIENO, ENTRAMBI I PIXEL vengono impostati al Multicolore 1)

COLORE ANIMAZIONE (Se il PIXEL DI SINISTRA di una coppia di pixel e' PIENO, ENTRAMBI I PIXEL vengono impostati al Colore Animazione)

MULTICOLORE 2 (Se ENTRAMBI I PIXEL di una coppia di pixel sono PIENI ENTRAMBI I PIXEL vengono impostati al Multicolore 2)

Consideriamo la riga di 8 pixel orizzontali illustrati sotto. Questo blocco imposta i primi due pixel al colore di fondo, i secondi due al multicolor 1, i terzi due al colore animazione ed i quarti due al multicolor 2. Il colore di ogni coppia di pixel dipende da quali bit di ogni coppia sono pieni e quali vuoti, secondo la precedente illustrazione. Dopo aver determinato i colori di ogni coppia di pixel bisogna aggiungere nel blocco di 8 pixel i valori dei pixel pieni e posizionare il risultato con una POKE nella locazione di memoria adatta. Ad esempio, se la riga di 8 pixel illustrata sotto e' il primo blocco di un'animazione che inizia alla locazione 832, il valore dei pixel pieni e' 16+8+2+1=27, percio' si deve impostare POKE832,27.

CONTATTI:

Si puo' scoprire se un'animazione e' entrata in collisione con un'altra usando questa istruzione:

IF PEEK(V+30)AND X=X THEN <AZIONE>.

Tale istruzione controlla se una data animazione e' entrata in collisione con un'altra animazione, dove X e' 1 per l'animazione 0, 2 per la 1, 4 per la 2,....,128 per la 7.

Per scoprire se l'animazione e' entrata in collisione con un "CARATTERE DI SOTTOFONDO", impostare:

IF PEEK(V+31)AND X=X THEN <AZIONE>

USO DEI CARATTERI GRAFICI NELLE ISTRUZIONI DATA

Il seguente programma permette di creare un'animazione usando spazi e cerchi pieni (SHIFT Q) nelle istruzioni data. L'animazione ed i numeri posizionati con una POKE nei registri dei dati dell'animazione vengono visualizzati.

```
        SHIFT CLR HOME
10 PRINT"J":FORI=0TO63:POKE832+I,0:NEXT
20 GOSUB60000
999 END
60000 DATA"         0000000        "
60001 DATA"        00000000000     "
60002 DATA"       0000000000000    "
60003 DATA"       00000    00000   "
60004 DATA"      00000 000   0000  "
60005 DATA"      00000 000   0000  "
60006 DATA"      00000 000   0000  "
60007 DATA"       00000    00000   "
60008 DATA"       0000000000000    "
60009 DATA"       0000000000000    "
60010 DATA"      0 000000000 0     "
60011 DATA"        0 0000000 0     "
60012 DATA"        0  00000  0     "
60013 DATA"         0  000  0      "
60014 DATA"         0  000  0      "
60015 DATA"          0  0  0       "
60016 DATA"          0  0  0       "
60017 DATA"            00000       "
60018 DATA"            00000       "
60019 DATA"            00000       "
60020 DATA"             000        "
60100 V=53248:POKEV,200:POKEV+1,100:POKEV+21,1:POKEV+39,14:POKE2040,13
60105 POKEV+23,1:POKEV+29,1
60110 FORI=0TO20:READA$:FORK=0TO2:T=0FORJ=0TO7:B=0
60140 IFMID$(A$,J+K*S+1,1)="O"THENB=1
60150 T=T+B*21(7-J):NEXT:PRINTT;:POKE832+I*3+K,T:NEXT:PRINT:NEXT
60200 RETURN
```

CAPITOLO 4

programmazione di suoni e musica con il commodore 64

- Introduzione
  Controllo del Volume
  Frequenze delle Onde Sonore
- Uso delle Voci Multiple
- Modifica delle Forme d'Onda
- Il Generatore di Inviluppo
- Filtratura
- Tecniche Avanzate
- Sincronizzazione e Modulazione Circolare

INTRODUZIONE

Il COMMODORE 64 e' dotato di uno dei piu' sofisticati sintetizzatori elettronici musicali disponibili su qualunque computer. Viene fornito completo di tre Voci totalmente indirizzabili, un Generatore ADSR (ATTACCARE/DECOMPORRE/SOSTENERE/RILASCIARE), filtratura, modulazione e "rumore bianco". Tutte queste capacita' sono messe a disposizione da poche istruzioni e funzioni del BASIC e/o del linguaggio assemblatore, facili da usare. Cio' significa che si possono comporre canzoni e suoni molto complessi usando programmi di progettazione relativamente semplice.

Questo Capitolo e' stato concepito per agevolare l'esplorazione di tutte le capacita' del Circuito 6581 "SID", il sintetizzatore sonoro e musicale. Vengono spiegate sia la teoria musicale, sia gli aspetti pratici che si incontrano nella conversione di tale teoria in composizioni reali complete.

Non e' necessario essere un programmatore o un musicista esperto per raggiungere con il sintetizzatore musicale dei risultati stimolanti. Questo Capitolo e' ricco di esempi di programmi, completi di spiegazione, da cui prendere spunto.

L'accesso al Generatore del Suono avviene tramite una POKE a particolari locazioni di memoria. La lista completa delle locazioni usate e' riportata nell'Appendice O. Ogni concetto viene illustrato passo dopo passo, in modo da mettere in condizione, verso la fine, di creare una varieta' di suoni quasi infinita, e di realizare esperimenti sonori in proprio.

Ogni paragrafo di questo Capitolo inizia dando un esempio di programma ed illustrandolo linea per linea, in modo da esporne le caratteristiche. La spiegazione tecnica e' a disposizione per tutte le occasioni in cui si desideri saperne di piu' su quanto sta accadendo.

L'istruzione fondamentale dei programmi musicali e' POKE; essa uguaglia la locazione di memoria indicata (MEM) ad un valore specificato (NUM):

```
POKE MEM,NUM
```

Le locazioni di memoria (MEM) usate per la sintesi musicale iniziano, sul COMMODORE 64, alla locazione 54272 ($D400 HEX). Le locazioni che vanno da 54272 a 54296, estremi inclusi, sono le locazioni da ricordare quando si fa uso della mappa registro del Circuito 6581 (SID). Un altro metodo per ricordare tali locazioni e' di fissare la sola locazione 54272, e poi di aggiungere un numero da 0 a 27. I numeri (NUM) da usare con l'istruzione POKE devono essere compresi fra 0 e 255.

Una volta presa pratica con la composizione della musica, si puo' approfondire la conoscenza ricorrendo alla funzione PEEK: questa e' una funzione che ritorna il valore corrente contenuto nella locazione di memoria indicata:

```
X=PEEK(MEM)
```

Il valore della variabile X e' posto uguale al contenuto corrente della locazione di memoria MEM.

Ovviamente, i programmi comprendono anche altre istruzioni BASIC, per la cui spiegazione si rimanda al Capitolo Istruzioni BASIC di questo Manuale.
Proviamo ora a scrivere un semplice programma usando una sola delle tre Voci a disposizione. Predisporre il computer, battere NEW, poi il programma seguente, ed infine RUN. Salvare quindi il programma su disco o su DATASSETTE (TM) Commodore.

ESEMPIO – PROGRAMMA 1:

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT:REM AZZERA IL CIRCUITO SONORO
20 POKES+5,9:POKES+6,0
30 POKES+24,15             :REM IMPOSTA IL VOLUME A MASSIMO
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4,32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

La seguente e' una descrizione linea per linea del programma appena battuto. Per quelle parti del programma non completamente comprese si rimanda a tale descrizione.

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 1 :

LINEA	DESCRIZIONE
5	Imposta S all'inizio del Circuito del Suono
10	Azzera tutti i registri del Circuito del Suono
20	Imposta ATTACCARE/DECOMPORRE per la Voce 1 (A=0, D=9) Imposta SOSTENERE/RILASCIARE per la Voce 1 (S=0, R=0)
30	Imposta il Volume al massimo
40	Legge l'Alta Frequenza, la Bassa Frequenza e la durata della Nota
50	Termina la canzone se Alta Frequenza < 0
60	Posiziona Alta e Bassa Frequenza della Voce 1
70	Introduce la Forma d'Onda "dente di sega" per la Voce 1.
80	Ciclo di tempo per la durata della nota
90	Rilascia la precedente Forma d'Onda
100	Posizionamento per la prossima nota
110-180	Dati della composizione: Alta Frequenza, Bassa Frequenza, Durata (numero di passi del ciclo) di ogni nota
190	Ultima nota della composizione e segnalazione di Termine Composizione (-1 sec)

CONTROLLO DEL VOLUME

Il registro 24 del circuito contiene l'intero controllo del Volume. Quest'ultimo puo' essere posizionato ad un qualunque valore compreso fra 0 e 15. Gli altri 4 bit sono usati per altri scopi che saranno definiti in seguito. La linea 30 del precedente programma illustra come il Volume viene impostato nel Programma 1.

FREQUENZE DELLE ONDE SONORE

Il suono e' generato in forma di onde dal movimento dell'aria. Se si getta un sasso in uno stagno, si possono osservare le onde che si allontanano a raggera dal punto dell'impatto; allo stesso modo, quando onde simili si creano in aria, siamo in grado di udirle. Se si misurano due successivi picchi d'onda, si trova il numero di secondi per ciclo dell'onda (n=numero di secondi). Il reciproco di questo numero (1/n) da' il numero di cicli per secondo; questa quantita' e' conosciuta anche come frequenza. L'acutezza o la profondita' di un suono (la nota) sono determinati dalla frequenza delle onde sonore prodotte.

Il Generatore del Suono del COMMODORE 64 usa due locazioni per determinare la frequenza; l'Appendice E riporta i valori delle frequenze necessari a riprodurre una gamma completa di otto ottave di note musicali. Per creare una frequenza piuttosto che un'altra elencata nella Tavola delle Note, si usi Fout (Frequency OUTput) e la formula seguente, per la rappresentazione della frequenza (Fn) del suono che si desidera creare. Va ricordato che ogni nota richiede un numero sia per l'Alta che per la Bassa Frequenza.

$$F_n = F_{out}/.06097$$

Stabilito il valore di Fn per la "nuova" nota, si tratta ora di creare, per quella nota, i valori di Alta e Bassa Frequenza. Innanzitutto, occorre arrotondare il valore di Fn ad un valore intero; a questo punto, il valore dell'Alta Frequenza e' dato da:

$$F_{hi} = F_n/256$$

mentre quello per la Bassa Frequenza e' dato da:

$$F_{lo} = F_n - (256 * F_{hi}).$$

USO DELLE VOCI MULTIPLE

Il COMMODORE 64 ha tre Voci (Oscillatori) controllate indipendentemente; il Programma 1 ne usava solo una. Piu' avanti, viene spiegato come cambiare la qualita' del suono prodotto dalle Voci.

Il programma seguente illustra come trasformare un foglio di carta in un'orchestra...computerizzata. Battere il seguente programma, quindi salvarlo su disco o su DATASSETTE (TM); ricordarsi di battere NEW prima di iniziare la battitura del programma.

ESEMPIO - PROGRAMMA 2:

```
10 S=54272:FORL=STOS+24:POKEL,0:NEXT
20 DIMH(2,200),L(2,200),C(2,200)
30 DIMFQ(11)
40 V(0)=17:V(1)=65:V(2)=33
```

ESEMPIO - PROGRAMMA 2:

```
50 POKES+10,8:POKES+22,128:POKES+23,244
60 FORI=0TO11:READFQ(I):NEXT
100 FORK=0TO2
110 I=0
120 READNM
130 IFNM=0THEN250
140 WA=V(K):IFNM<0THENNM=-NM:WA=1
150 DR%=NM/128:OC%=(NM-128*DR%)/16
160 NT=NM-128*DR%-16*OC%
170 FR=FQ(NT)
180 IFOC%=7THEN200
190 FORJ=6TOOC%STEP-1:FR=FR/2:NEXT
200 HF%=FR/256:LF%=FR-256*HF%
210 IFDR%=1THENH(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:GOTO120
220 FORJ=1TODR%-1:H(K,I)=HF%:L(K,I)=LF%:C(K,I)=WA:I=I+1:NEXT
230 H(K,I)=HF%:L(K,I=LF%:C(K,I)=WA-1
240 I=I+1:GOTO120
250 IFI>IMTHENIM=I
260 NEXT
500 POKES+5,0:POKES+6,240
510 POKES+12,85:POKES+13,133
520 POKES+19,10:POKES+20,197
530 POKES+24,31
540 FORI=0TOIM
550 POKES,L(0,I):POKES+7,L(I,1):POKES,L(2,1)
560 POKES+1,H(0,I):POKES+8,H(1,I):POKES+15,H(2,I)
570 POKES+4,C(0,1):POKES+11,C(1,1):POKES+18,H(2,I)
580 FORT=1TO80:NEXT:NEXT
590 FORT=1TO200:NEXT:POKES+24,0
600 DATA34334,36376,38539,40830
610 DATA43258,45830,48556,51443
620 DATA54502,57743,61176,64814
1000 DATA594,594,594,596,596
1010 DATA1618,587,592,587,585,331,336
1020 DATA1097,583,585,585,585,587,587
1030 DATA1609,585,331,337,594,594,593
1040 DATA1618,594,596,594,592,587
1050 DATA1616,587,585,331,336,841,327
1060 DATA1607
1999 DATA0
2000 DATA583,585,583,583,327,329
2010 DATA1611,583,585,578,578,578
2020 DATA196,198,583,326,578
2030 DATA326,327,329,327,327,329,326,578,583
2040 DATA1606,582,322,324,582,587
2050 DATA329,327,1606,583
2060 DATA327,329,587,331,329
2070 DATA329,328,1609,578,834
2080 DATA324,322,327,585,1602
2999 DATA0
3000 DATA567,566,567,304,306,308,310
3010 DATA1591,567,311,310,567
3020 DATA306,304,299,308
3030 DATA304,171,176,306,291,551,306,308
3040 DATA310,308,310,306,295,297,299,304
3050 DATA1586,562,567,310,315,311
3060 DATA308,313,297
3070 DATA1586,567,560,311,309
3080 DATA308,309,306,308
3090 DATA1577,399,295,306,310,311,304
3100 DATA562,546,1575
3999 DATA0
```

La seguente e' una spiegazione linea per linea del Programma 2; ci occuperemo, per ora, di come sono controllate le tre Voci.

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 2:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del Circuito del Suono e azzera tutti i registri di tale Circuito
20	Dimensiona le schiere per contenere l'attivita' della composizione, 1/16 di tempo per locazione
30	Dimensiona le schiere per contenere la Frequenza di Base di ogni nota
40	Registra il byte di controllo della Forma d'Onda per ogni Voce
50	Imposta l'ampiezza dell'impulso alto per la Voce 2 Imposta l'Alta Frequenza per il taglio del filtro Imposta la risonanza per il filtro e la Voce 3 del filtro
60	Legge la frequenza base di ogni nota
100	Inizio del ciclo di decodifica di ogni Voce
110	Inizializza il puntatore alla schiera attivita'
120	Legge la nota codificata
130	Se questa e' zero passa alla Voce successiva
140	Imposta il controllo della forma d'onda alla Voce appropriata
150	Decodifica durata ed ottava
160	Decodifica la nota
170	Acquisisce la frequenza base di questa nota
180	Se e' l'ottava piu' alta, salta il ciclo di divisione
190	Divide la frequenza base per due appropriate quantita' di tempo
200	Acquisisce i bytes di Alta e Bassa Frequenza
210	Se e' la sedicesima nota, imposta la schiera attivita': Alta Frequenza, Bassa Frequenza e controllo della forma d'onda (Voce ON)
220	Per tutte le battute meno l'ultima imposta la schiera attivita': alto
230	Per l'ultima battuta, imposta la schiera attivita' : Alta Frequenza, Bassa Frequenza, controllo della forma d'onda (Voce OFF)
240	Incrementa il puntatore alla schiera attivita' Acquisisce la prossima nota
250	Se e' piu' lunga della precedente, imposta daccapo il numero delle attivita'
260	Ritorna alla prossima Voce
500	Imposta ATTACCARE/DECOMPORRE per la Voce 1 (A=0, D=0)
510	Come sopra, ma per la Voce 2 (A=5, D=5, S=8, R=5)
520	Come 500, ma per la Voce 3 (A=0, D=10, S=12, R=5)
530	Imposta il Volume a 15 ed il filtro passabasso
540	Inizio del ciclo per ogni 1/16 di tempo
550	Preleva (POKE) la Bassa Frequenza dalla schiera delle attivita' per tutte le Voci
560	Come sopra, ma per l'Alta Frequenza
570	Come 550 ma per la forma d'onda
580	Ciclo di tempo per 1/16 di tempo e ritorno per il prossimo
590	Pausa, poi disattiva il Volume
600-620	Dati della frequenza base
1000-1999	Dati della Voce 1
2000-2999	Dati della Voce 2
3000-3999	dati della Voce 3

I valori usati nelle istruzioni DATA sono stati calcolati usando la Tabella delle Note (Appendice E) e la Tavola seguente:

TIPO DI NOTA	DURATA
1/16	128
1/8	256
1/8 Punteggiato	384
1/4	512
1/4 + 1/16	640
1/4 Punteggiato	768
1/2	1024
1/2 + 1/16	1152
1/2 + 1/8	1280
1/2 Punteggiato	1536
Intero	2048

Il numero della nota preso dalla Tabella delle Note viene sommato alla precedente durata. Successivamente, si puo' introdurre ogni nota usando solamente un numero decodificato dal programma. La formula usata per codificare una nota e' la seguente:

1) Si moltiplica la durata (numero di sedicesimi del tempo) per 8
2) Si aggiunge al risultato l'ottava che si e' scelto (0-7)
3) Si moltiplica il risultato per 16
4) Si aggiunge al risultato la nota che si e' scelto

In altri termini:

$$((((D*8)+O)*16)+N)$$

essendo D=Durata, O=Ottava, N=Nota.
Il silenzio si ottiene usando il negativo della quantita' della durata (numero di sedicesimi di tempo X 128) (Questo e' soltanto uno dei possibili metodi di codifica, fra i quali si puo' scegliere quello individualmente piu' confacente).

CONTROLLO DELLE VOCI MULTIPLE

Dopo aver imparato ad usare piu' di una Voce, si puo' osservare che e' necessario coordinare il ritmo delle tre Voci. Nel programma precedente, cio' viene realizzato:

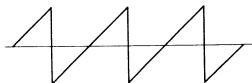
1) Dividendo ogni tempo musicale in 16 parti
2) Memorizzando in tre schiere separate gli eventi che accadono ad ogni intervallo di sedicesimo di tempo

I bytes di Alta e Bassa Frequenza sono calcolati dividendo per 2 le frequenze dell'ottava piu' alta (linee 180 e 190). Il byte di controllo della forma d'onda e' unsegnale di partenza per iniziare una nota o continuarne un'altra che sta gia' suonando; analogamente, lo stesso byte viene usato come segnale di fine nota. La scelta della forma d'onda viene effettuata, per ogni Voce, alla linea 40.
Questo e' solamente uno dei possibili metodi di controllo delle Voci multiple, fra i quali si puo' scegliere quello individualmente piu' confacente; quello che importa, tuttavia, e' di essere in grado di

rappresentare le note per tutte le tre Voci.

MODIFICA DELLE FORME D'ONDA

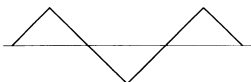
La qualita' tonale di un suono si chiama TIMBRO. Il timbro di un suono e' determinato essenzialmente dalla sua "forma d'onda". Se si ricorda l'esempio del sasso gettato in acqua, si ricordera' anche che le onde si propagano uniformemente sullo stagno. Queste onde assomigliano alquanto alla prima onda sonora di cui stiamo per parlare: l'onda sinusoidale (illustrata sotto).

Per rendere un po' piu' pratico l'argomento di cui stiamo parlando, ritorniamo al Programma 1 per esaminare differenti forme d'onda, in quanto le modifiche che stiamo per apportare sono piu' facili se si usa una sola Voce. Questo programma usa una forma d'onda a "dente di sega" come la seguente:

che gli deriva dal Dispositivo Generatore del Suono del Circuito 6581 SID. Se si cambia il numero di partenza della nota nella linea 70 da 33 a 17, ed il corrispondente numero di arrivo nella linea 90 da 32 a 16, si ottiene il seguente programma:

ESEMPIO - PROGRAMMA 3 (PROGRAMMA 1 MODIFICATO):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,9:POKES+6,0
30 POKES+24,15
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,17
80 FORT=1TODR:NEXT
```

```
90 POKES+4,16:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Se ora lanciamo (RUN) il programma, notiamo che la qualita' del suono e' diversa, meno stridula e piu' cupa: cio' poiche' abbiamo cambiato la forma d'onda da "dente di sega" a triangolare.

La terza forma d'onda e' detta ad impulso variabile:

Come illustrato in figura, questa e' un'onda rettangolare di cui si deve determinare la lunghezza del ciclo di pulsazione, definendo la proporzione della parte alta dell'onda. Cio' e' ottenuto, per la Voce 1, usando i registri 2 e 3: il registro 2 e' il byte basso dell'ampiezza dell'impulso (Lpw=0...255); il registro 3 sono i quattro bit alti (Hpw=0...15).
Questa coppia di registri specifica un numero a 12 bit per l'ampiezza dell'impulso; tale numero puo' essere determinato dalla seguente formula:

$$PW_n = H_{pw}*256 + L_{pw}$$

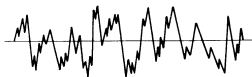
mentre l'ampiezza del suono e' determinata dalla seguente relazione:

$$PW_{out} = (PW_n/40.95)\ \%$$

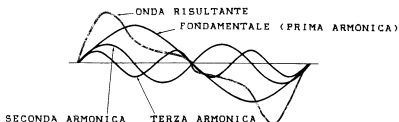
Quando PWn assume il valore 2048, l'onda assume una forma quadrata:

cio' vuol dire che il registro 2 (Lpw)=0 ed il registro 3 (Hpw)=8.
Aggiungiamo ora al programma in questione la seguente riga:

```
15 POKES+3,8:POKES+2,0
```

e cambiamo il numero di inizio nella linea 70 in 65, e quello di fine nella linea 90 in 64; lanciamo (RUN) infine il programma. Se a questo punto si modifica l'ampiezza della pulsazione alta (registro 3 alla linea 15) da 8 a 1, si nota una differenza di suono...drammatica: quest'ultima forma d'onda e' il rumore bianco:

Questa forma d'onda e' usata soprattutto per effetti sonori. Per udire questo suono, occorre cambiare il numero di inizio nella linea 70 in 129, ed il numero di fine nella linea 90 in 128.

INTRODUZIONE ALLE FORME D'ONDA

Una nota che viene suonata e' formata da un'onda sinusoidale, oscillante alla frequenza fondamentale, e dalle armoniche di quell'onda.
La frequenza fondamentale definisce completamente la tonalita' della nota. Le armoniche sono onde sinusoidali la cui frequenza e' un multiplo intero della frequenza fondamentale. Un'onda sonora e' composta dalla frequenza fondamentale e da tutte le armoniche richieste per formare quel suono.

La teoria musicale assume l'armonica numero 1 come frequenza fondamentale; la seconda armonica ha una frequenza doppia di quella fondamentale, la terza armonica tripla, ecc. La quantita' di ogni armonica presente in una nota da' il timbro della nota stessa.
Uno strumento acustico, come una chitarra o un violino, ha una struttura armonica molto complessa, per il fatto che tale struttura puo' variare a seconda di come viene variata una singola nota. Le forme d'onda disponibili al sintetizzatore musicale del COMMODORE 64 sono gia' state esaminate; rimane da affrontare il problema di come

funzionano le armoniche con onde triangolari, rettangolari ed a "dente di sega".
Un'onda triangolare possiede soltanto armoniche casuali; la quantita' di ogni armonica presente e' proporzionale al reciproco del quadrato del numero di armonica. In altre parole, l'armonica numero 3 e' 1/9 piu' dolce dell'armonica numero 1, in quanto il quadrato di 3 e' 9 (3 X 3 =9), ed il suo reciproco e' 1/9.
Come si puo' osservare, c'e' una somiglianza nella forma di un'onda triangolare rispetto ad un'onda sinusoidale oscillante alla frequenza fondamentale.
Un'onda a "dente di sega" contiene tutte le armoniche; la quantita' di ogni armonica presente e' proporzionale al reciproco del numero di armonica. Ad esempio, l'armonica numero 2 e' profonda 1/2 rispetto all'armonica numero 1.
L'onda rettangolare contiene armoniche casuali in proporzione al reciproco del numero di armonica. Onde rettangolari diverse hanno un diverso contenuto armonico. Cambiando l'ampiezza dell'impulso, viene modificato grandemente il timbro del suono di un'onda rettangolare.
Scegliendo attentamente la forma d'onda usata, si puo' dare inizio ad una struttura armonica che assomiglia in qualche modo al suono che si desidera riprodurre. Per la rifinitura di tale suono, si puo' aggiungere un'altra caratteristica della qualita' del suono disponibile sul COMMODORE 64, chiamata filtratura, della quale parleremo piu' avanti.

IL GENERATORE DI INVILUPPO

Il volume di un tono musicale cambia dal momento in cui viene percepito, via via fino alla sua scomparsa, quando non puo' piu' essere udito. Quando una nota viene suonata per la prima volta, il suo volume sale da zero al suo volume di picco. Il passo in cui cio' si verifica si chiama ATTACCARE. Successivamente, la nota scende di volume dal valore di picco ad un valore medio: questo passo prende il nome di DECOMPORRE, mentre il livello medio raggiunto si chiama SOSTENERE. Quando infine la nota cessa di suonare, il Volume passa dal livello di SOSTENERE al valore zero: il passo in cui cio' si verifica si chiama RILASCIARE. Una rappresentazione di queste quatro fasi e' data qui sotto:

Ognuno dei quattro livelli sopra menzionati conferisce ad una nota certe qualita' e restrizioni; i quattro livelli si chiamano parametri, e vengono indicati collettivamente con le rispettive iniziali (ADSR); il loro controllo avviene per mezzo di un insieme di locazioni del Circuito Generatore del Suono. Riprendiamo il primo programma di questo capitolo, carichiamolo e lanciamolo cercando di ricordarsi il suono emesso. A questo punto, cambiamo la linea 20 in modo che il

programma diventi come il seguente:

ESEMPIO - PROGRAMMA 4 (PROGRAMMA 1 MODIFICATO):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
20 POKES+5,88:POKES+6,195
30 POKES+24,15
40 REDAHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF
70 POKES+4,33
80 FORT=1TODR:NEXT
90 POKES+4.32:FORT=1TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,25,24,63,125
190 DATA19,63,250,-1,-1,-1
```

I registri 5 e 6 definiscono l'ADSR per la Voce 1: l'ATTACCARE e' il semibyte alto del registro 5 (si intende per semibyte l'insieme delle quattro locazioni (bit) piu' alte o piu' basse di un registro), mentre il DECOMPORRE e' il semibyte basso. Per l'ATTACCARE si puo' scegliere qualunque numero compreso fra 0 e 15; moltiplicandolo poi per 16 ed aggiungendo un numero compreso fra 0 e 15 si ottiene il DECOMPORRE. I valori che corrispondono a questi numeri sono elencati piu' in basso.
Il livello SOSTENERE e' il semibyte alto del registro 6; e' compreso fra 0 e 15; definisce la quantita' del volume di picco posseduta. Il RILASCIARE e' il semibyte basso del registro 6.

VALORE	VALORE DI ATTACCARE (TEMPO/CICLO)	VALORE DI DECOMPORRE/RILASCIARE (TEMPO/CICLO)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

I seguenti sono solo pochi esempi di modifiche da apportare al precedente programma, che insieme alle modifiche realizzabili individualmente forniscono una varieta' di suoni veramente sbalorditiva! Se, ad esempio, si vuole riprodurre il suono di un violino, occorre modificare la linea 20 come segue:

```
20 POKES+5,88:POKES+6,89:REM A=5;D=8;S=5;R=9
```

mentre uno xilofono sara' riprodotto modificando in triangolare la forma d'onda, ed introducendo le seguenti linee:

```
20 POKES+5,9:POKES+6,9:REM A=0;D=9;S=0;R=9     S=0;R=9
70 POKES+4,17
90 POKES+4,16: FORT=1TO50:NEXT
```

Se poi si usa un'onda rettangolare si potra' ottenere il suono di un pianoforte impostando:

```
15 POKES+3,8:POKES+2,0
20 POKES+5,9:POKES+6,0: REM A=0;D=9;S=0;R=0
70 POKES+4,65
90 POKES+4,64:FORT=1TO50:NEXT
```

Ma i suoni piu' emozionanti sono quelli che solo il sintetizzatore possiede e che non imitano alcuno strumento acustico; ad esempio:

```
20 POKES+5,144:POKES+6,243:REM A=9;D=0; S=15;R=3
```

FILTRATURA

Il contenuto sonoro di una forma d'onda puo' essere modificato usando un filtro. Il circuito SID e' equipaggiato con tre tipi di filtri, che possono essere usati singolarmente o in combinazione. Come dimostrazione dell'uso di un filtro, torniamo a considerare il PROGRAMMA 1: ci sono diversi controlli di filtro da impostare.

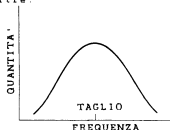
Per impostare la FREQUENZA DI TAGLIO del filtro aggiungere la linea 15; la frequenza di taglio e' il punto di riferimento del filtro. I punti delle frequenze di taglio alte e basse vengono IMPOSTATE nei registri 21 e 22. L'attivazione del filtro per la Voce 1 avviene caricando (POKE) il registro 23.
Successivamente, per mostrare che si usa un filtro passa alto occorre modificare la linea 30 (vedere la mappa dei registri del SID).

ESEMPIO - PROGRAMMA 5 (PROGRAMMA 1 MODIFICATO):

```
5 S=54272
10 FORL=STOS+24:POKEL,0:NEXT
15 POKES+2,128:POKES+21,0:POKES+23,1
20 POKES+5,9:POKES+6,0
30 POKES+24,79
40 READHF,LF,DR
50 IFHF<0THENEND
60 POKES+1,HF:POKES,LF
70 POKES+4,33
80 FORT1TODR:NEXT
90 POKES+24,32:FORT1=TO50:NEXT
100 GOTO40
110 DATA25,177,250,28,214,250
120 DATA25,177,250,25,177,250
130 DATA25,177,125,28,214,125
140 DATA32,94,750,25,177,250
150 DATA28,214,250,19,63,250
160 DATA19,63,250,19,63,250
170 DATA21,154,63,24,63,63
180 DATA25,177,250,24,63,125
190 DATA19,63,250,-1,-1,-1
```

Se ora si prova a lanciare il programma, si notera' che i toni bassi hanno un volume ridotto; cio' provoca un cambiamento della qualita' complessiva del suono della nota, che ora sembrera' piu' metallica: infatti, il filtro passa alto usato attenua (riduce) le frequenze al disotto della frequenza di taglio usata.
Il circuito SID del COMMODORE 64 possiede tre tipi di filtri; abbiamo appena usato il filtro passa alto, che lascia passare tutte le freguenze maggiori o uguali a quella di taglio, mentre attenua le frequenze al di sotto di quella di taglio.

Il circuito SID possiede anche un filtro passa basso, che lascia passare le frequenze al di sotto di quelle di taglio ed attenua quelle al di sopra.

Infine si ha a disposizione un filtro passa banda, che lascia passare una banda di frequenze ristrette intorno alla frequenza di taglio, attenuando tutte le altre.

I filtri passa alto e passa basso possono essere combinati per formare un filtro di rigetto del taglio, che attenua la frequenza di taglio lasciando passare tutte le altre.

Il registro 24 determina il tipo di filtro usato: questo in aggiunta alle altre funzioni di questo registro, come il controllo completo del volume. Il bit 6 controlla il filtro passa alto (0=OFF, 1=ON), il bit 5 il filtro passa banda e il bit 4 il filtro passa basso. I tre bit bassi della frequenza di taglio sono determinati dal registro 21 (Lcf=0...7), mentre il registro 22 determina gli 8 BIT della frequenza di taglio alta (Hcf=0...255). Per mezzo di un uso oculato di filtri, si puo' cambiare la struttura armonica di qualunque forma d'onda, ottenendo cosi' il suono desiderato. Inoltre, si possono produrre effetti interessanti cambiando la filtratura di un suono durante il suo movimento attraverso le 4 fasi ADSR.

TECNICHE AVANZATE

I parametri del circuito SID possono essere modificati dinamicamente durante la riproduzione di una nota o di un suono, creando cosi' molti effetti interessanti e divertenti. Allo scopo di facilitare la loro messa in paratica, sono disponibili, rispettivamente nei registri 27 e 28, le uscite digitalizzate provenienti dall'oscillatore 3 e dal generatore di inviluppo 3. L'uscita dell'oscillatore 3 (registro 27) e' collegata direttamente alla forma d'onda scelta. Se ad esempio si sceglie la forma d'onda a "dente di sega", questo registro presenta una serie di numeri crescenti da 0 a 255 secondo un incremento determinato dalla frequenza dell'oscillatore 3. Se invece si sceglie una forma d'onda triangolare, l'uscita viene prima incrementata da 0 a 255, poi decrementata da 255 a 0. Se, infine, si sceglie la forma d'onda di un rumore, si ottiene una serie di numeri casuali. Quando

l'oscillatore 3 viene usato per la modulazione, di solito NON si desidera udire la sua uscita: cio' e' possibile impostando a 0 (OFF) il bit 7 del registro 24, che esclude l'OUTPUT audio della voce 3. Il registro 27 riporta sempre l'uscita modificata dell'oscillatore; questo registro non viene influenzato in alcun modo dal generatore d'inviluppo (ADSR).

L'accesso all'uscita del generatore d'inviluppo dell'oscillatore 3 e' data dal registro 25, che funziona quali allo stesso modo dell'uscita dell'oscillatore 3. Questo oscillatore deve essere attivato per produrre qualunque uscita da questo registro.

L'effetto "vibrato" (una rapida variazione di frequenza) puo' essere ottenuto aggiungendo l'uscita dell'oscillatore 3 alla frequenza di un'altro oscillatore; questa idea e' illustrata nel programma 6.

ESEMPIO - PROGRAMMA 6:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+3,8
40 POKES+5,41:POKES+6,89
50 POKES+14,117
60 POKES+18,16
70 POKES+24,143
80 READFR,DR
90 IFFR=0THENEND
100 POKES+4,65
110 FORT=1TODR*2
120 FQ=FR+PEEK(S+27)/2
130 HF=INT(FQ/256):LF=FQAND255
140 POKES+0,LF:POKES+1,HF
150 NEXT
160 POKES+4,64
170 GOTO80
500 DATA4817,2,5103,2,5407,2
510 DATA8583,4,5407,2,8583,4
520 DATA5407,4,8583,12,9634,2
530 DATA10207,2,10814,2,8583,2
540 DATA9634,4,10814,2,8583,2
550 DATA9634,4,8583,12
560 DATA0,0
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 6:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera tutte le locazioni del circuito del suono
30	Imposta la voce 1 ad alta ampiezza di pulsazione
40	Imposta ATTACCARE/DECOMPORRE per questa voce (A=2, D=9) Imposta SOSTENERE/RILASCIARE per questa voce (S=5,R=9)
50	Imposta la voce 3 a bassa frequenza
60	Imposta la forma d'onda triangolare per questa voce
70	Imposta il volume a 15; disattiva l'uscita audio della voce 3
80	Legge frequenze e durata della nota
90	Se tale frequenza e' 0 si ferma
100	Imposta all'inizio il controllo della pulsazione della forma d'onda della voce 1
110	Inizio del ciclo di tempo della battuta
120	Acquisisce una nuova frequenza usando l'uscita dell'oscillatore 3
130	Acquisisce alta e bassa frequenza
140	Imposta alta e bassa frequenza per la voce 1
150	Fine del ciclo di tempo della battuta
160	Imposta alla fine il controllo della pulsazione della forma d'onda della voce 1
170	Passa alla prossima nota.
500-550	Frequenze e battute della composizione
560	Segnale di fine composizione

Una vasta gamma di effetti sonori puo' essere ottenuta anche per mezzo di effetti dinamici. Ad esempio, il seguente programma, che

realizza l'ululato di una sirena, modifica dinamicamente l'uscita della frequenza dell'oscillatore 1, quando quest'ultimo sia stato regolato sull'uscita dell'onda triangolare dell'oscillatore 3:

ESEMPIO - PROGRAMMA 7:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+14,5
40 POKES+18,16
50 POKES+3,1
60 POKES+24,143
70 POKES+6,240
80 POKES+4,65
90 FR=5389
100 FORT=1TO200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKES+0,LF:POKES+1,HF
140 NEXT
150 POKES+24,0
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 7:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta la bassa frequenza della voce 3
40	Imposta la forma d'onda triangolare per questa voce
50	Imposta la voce 1 ad alta ampiezza di pulsazioni
60	Imposta il volume a 15 e disattiva l'uscita audio
70	Imposta SOSTENERE/RILASCIARE per la voce 1 (S=15, R=0)
80	Imposta all'inizio il controllo della pulsazione della forma d'onda della voce 1
90	Imposta la frequenza piu' bassa per la sirena
100	Inizio del ciclo di tempo
110	Aquisisce una nuova frequenza usando l'uscita dell'oscillatore 3
120	Aquisisce alta e bassa freguenza
130	Imposta alta e bassa frequenza per la voce 1
140	Fine del ciclo di tempo
150	Disattiva il volume

La forma d'onda del rumore puo' essere usata per fornire una varieta' di effetti sonori. Questo esempio imita un applauso usando una forma d'onda di rumore filtrato:

ESEMPIO – PROGRAMMA 8:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+0,240:POKES+1,33
40 POKES+5,8
50 POKES+22,104
60 POKES+23,1
70 POKES+24,79
80 FORN1TO15
90 POKES+4,129
100 FORT=1TO250:NEXT:POKES+4,128
110 FORT=1TO30:NEXT:NEXT
120 POKES+24,0
```

DESCRIZIONE LINEA PER LINEA DEL PROGRAMMA 8:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta alta e bassa frequenza per la voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa voce (A=0, D=8)
50	Imposta l'alta frequenza di taglio per il filtro
60	Attiva il filtro per la voce 1
70	Imposta il volume a 15 del filtro passa alto
80	Conta 15 applausi
90	Imposta all'inizio il controllo della forma d'onda rumore
100	Attesa, poi imposta alla fine il controllo della forma d'onda del rumore
110	Attesa, poi inizia il prossimo applauso
120	Disattiva il volume

SINCRONIZZAZIONE E MODULAZIONE CIRCOLARE

Il circuito 6581 SID permette di creare strutture armoniche piu' complesse per mezzo della sincronizzazione o modulazione circolare di due Voci.

Il processo di sincronizzazione consiste fondamentalmente in una AND logica di due forme d'onda; quando entrambi sono zero, il risultato e' nullo. L'esempio seguente usa questo processo per imitare una zanzara:

ESEMPIO – PROGRAMMA 9:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,100
40 POKES+5,219
50 POKES+15,28
60 POKES+24,15
70 POKES+4,19
80 FORT=1TO5000:NEXT
90 POKES+4,18
100 FORT=1TO1000:NEXT:POKES+24,0
```

DESCRIZIONE LINEA PER LINEA DEL PROGRAMMA 9:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circuito del suono
30	Imposta ad alta frequenza la Voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa Voce (A=13, D=11)
50	Imposta ad alta frequenza la Voce 3
60	Imposta il Volume a 15
70	Imposta all'inizio il controllo del sincronismo e della forma d'onda triangolare della Voce 1
80	Ciclo di tempo
90	Imposta alla fine il controllo del sincronismo e della forma d'onda triangolare della Voce 1
100	Attesa, poi disattiva il Volume

La caratteristica della sincronizzazione viene attivata alla linea 70, dove vengono impostati i bit 0, 1 e 4 del registro 4. Il bit 1 attiva la funzione di sincronizzazione tra le Voci 1 e 3; i bit 0 e 4 svolgono le loro funzioni abituali di introduzione della Voce 1 e di impostazione della forma d'onda triangolare.

La modulazione circolare (realizzata, per la Voce 1, impostando a 1 il bit 3 del registro 4 nella linea 70 del programma sseguente) sostituisce l'uscita triangolare dell'oscillatore 1 con una combinazione "modulata ad anello" degli oscillatori 1 e 3. Cio' produce strutture sopratonali non armoniche che trovano impiego nell'imitazione del suono di campane o di gong. Il seguente programma esegue l'imitazione di una sveglia:

ESEMPIO - PROGRAMMA 10:

```
10 S=54272
20 FORL=0TO24:POKES+L,0:NEXT
30 POKES+1,130
40 POKES+5,9
50 POKES+15,30
60 POKES+24,15
70 FORL=1TO12:POKES+4,21
80 FORT=1TO1000:NEXT:POKES+4,20
90 FORT=1TO1000:NEXT:NEXT
```

SPIEGAZIONE LINEA PER LINEA DEL PROGRAMMA 10:

LINEA	DESCRIZIONE
10	Imposta S all'inizio del circuito del suono
20	Azzera i registri del circùito del suono
30	Imposta l'alta frequenza per la Voce 1
40	Imposta ATTACCARE/DECOMPORRE per questa Voce (A=0, D=9)
50	Imposta l'alta frequenza per la Voce 3
60	Imposta il Volume a 15
70	Conta il numero di rintocchi ed imposta all'inizio il controllo della forma d'onda triangolare e della modulazione circolare
80	Ciclo di tempo; imposta alla fine la forma d'onda triangolare e la modulazione circolare
90	Ciclo di tempo; rintocco successivo

Con l'uso dei parametri del circuito SID del COMMODORE 64 si hanno a disposizione effetti numerosi e vari. Solamente con delle prove personali si possono apprezzare appieno le capacita' della macchina: gli esempi dati in questo Capitolo hanno un valore puramente indicativo.

Per avere una conoscenza di tipo piu' professionale, svincolata dal gioco e dal semplice divertimento, si rimanda al testo MAKING MUSIC ON YOUR COMMODORE COMPUTER.

CAPITOLO 5

dal basic al linguaggio macchina

- Che cos'è un Linguaggio Macchina?
- Come si scrivono i programmi in Linguaggio Macchina?
- Notazione Esadecimale
- Modi di Indirizzamento
- Indicizzazione
- Sottoprocedure
- Suggerimenti utili per il Principiante
- Un compito più impegnativo
- Insieme delle istruzioni del microprocessore MCS6510
- Gestione della memoria sul Commodore 64
- Il KERNAL
- Attività di inizializzazione del KERNAL
- Uso del Linguaggio Macchina da BASIC
- Mappa della memoria del Commodore 64

CHE COS'È IL LINGUAGGIO MACCHINA?

Il cuore di ogni microcomputer e' costituito da un microprocessore centrale, un particolare microcircuito (chip) che rappresenta il cervello del computer. Il COMMODORE 64 non fa eccezione. Ciascun microprocessore comprende le istruzioni scritte nel proprio linguaggio. Per essere piu' precisi, il Linguaggio Macchina e' il solo linguaggio di programmazione che il COMMODORE 64 comprenda. Esso e' il linguaggio NATURALE della macchina.

Se il Linguaggio Macchina e' il solo linguaggio che il COMMODORE 64 comprende, allora come puo' capire il linguaggio di programmazione CBM BASIC ? Il CBM BASIC NON e' il Linguaggio Macchina del COMMODORE 64. Che cosa, quindi, fa si' che il COMMODORE 64 capisca istruzioni CBM BASIC come PRINT o GOTO ?

Per avere una risposta, si deve vedere per prima cosa, cio' che accade dentro il COMMODORE 64. A prescindere dal microprocessore, che e' il cervello del COMMODORE 64, c'e' un programma in Linguaggio Macchina che e' memorizzato in uno speciale tipo di memoria in modo tale da non poter essere modificato. E, cosa ancora piu' importante, esso non viene perduto quando il COMMODORE 64 viene spento, diversamente da quanto accade ad un programma scritto da un Utente. Questo programma in Linguaggio Macchina e' chiamato SISTEMA OPERATIVO del COMMODORE 64. Il COMMODORE 64 sa che cosa fare quando e' acceso perche' il suo Sistema Operativo "gira" automaticamente.

Il Sistema Operativo e' incaricato di "organizzare" tutta la memoria della macchina affinche' svolga varie mansioni. Oltre ad un certo numero di altre funzioni, esso considera inoltre quali caratteri vengono digitati sulla tastiera e li riporta sullo schermo. Il Sistema Operativo si puo' pensare come "l'intelligenza e la personalita'" del COMMODORE 64 (o di ogni altro computer di quella portata). Cosi', quando si accende il COMMODORE 64, il Sistema Operativo prende il controllo della macchina, e dopo aver terminato il suo compito, dice:

```
READY.
■
```

Il Sistema Operativo del COMMODORE 64 permette, quindi, di digitare sulla tastiera e di usare l'EDITOR SCHERMO del COMMODORE 64. L'Editor di Schermo permette di muovere il cursore, di cancellare (DEL), di immettere (INS), ecc., ed e', percio', l'unica parte del Sistema Operativo incorporata a vantaggio dell'Utente.

Tutti i comandi che sono disponibili nel CBM BASIC sono semplicemente riconosciuti da un altro vasto programma in linguaggio macchina incorporato nel COMMODORE 64. Questo vasto programma "elabora" (RUN) il segmento appropriato del linguaggio macchina collegato al comando in CBM BASIC che sta per essere eseguito. Questo programma e' chiamato l'INTERPRETE BASIC, perche' interpreta ciascun comando, uno ad uno, a meno che incontri un comando che non capisce, nel qual caso appare il familiare messaggio di:

```
?SYNTAX ERROR

READY.
■
```

A CHE COSA ASSOMIGLIA IL CODICE MACCHINA?

Si dovrebbe avere familiarita' con i comandi PEEK e POKE del linguaggio CBM BASIC, con i quali si modificano le locazioni di memoria. Probabilmente, sono gia' stati usati per la grafica sullo schermo e per gli effetti sonori. Ciascuna locazione di memoria ha il proprio numero di identificazione. Tale numero e' conosciuto come l'"indirizzo" di una locazione di memoria. Se si immagina la memoria del COMMODORE 64 come una strada circondata da edifici, allora il numero su ciascuna porta e', ovviamente, l'indirizzo. Vediamo ora per quali scopi vengono utilizzate le varie parti della strada.

SEMPLICE MAPPA DELLA MEMORIA DEL COMMODORE 64

INDIRIZZO	DESCRIZIONE
0 & 1	Registri del 6510
2	Inizio della memoria
Fino a: 1023	Memoria usata dal Sistema Operativo
Da 1024 a 2039	Memoria dello schermo
Da 2040 a 2047	Puntatori degli sprites
Da 2048 a 40959	Memoria UTENTE. Qui vengono memorizzati il BASIC ed i programmi in Linguaggio Macchina (o entrambi)
Da 40960 a 49151	Interprete CBM BASIC (8K)
Da 49152 a 53247	Area RAM per programmi speciali
Da 53248 a 53294	Registri del VIC-II
Da 55296 a 56296	RAM colore
Da 56320 a 57343	Registri di I/O
Da 57344 a 65535	Sistema Operativo CBM KERNAL (8K)

Se non si comprende il significato della descrizione appena data di ciascuna parte della memoria, risultera' piu' chiaro in altre parti di questo Manuale. I programmi in Linguaggio Macchina sono costituiti da istruzioni che possono o no avere operandi (parametri) associati ad essi. Ciascuna istruzione occupa una locazione di memoria, ed ogni operando e' contenuto in una o due locazioni contigue all'istruzione. Nei programmi in BASIC, parole come PRINT e GOTO vanno ad occupare solamente una locazione di memoria, invece che una per ogni carattere della parola. I contenuti della locazione che rappresenta una particolare parola riservata del BASIC e' chiamato "token". Nel Linguaggio Macchina, ci sono "token" diversi per differenti istruzioni, anch'essi occupanti un solo byte (locazione di memoria = byte).

Le istruzioni del Linguaggio Macchina sono molto semplici. Percio', ciascuna istruzione individuale non puo' realizzare moltissimo. Le istruzioni in Linguaggio Macchina modificano il contenuto di una locazione di memoria, oppure cambiano uno dei registri interni

(particolari locazioni di memoria) del microprocessore. I registri interni rappresentano la vera base del linguaggio macchina.

I REGISTRI DEL MICROPROCESSORE 6510

ACCUMULATORE

Questo e' il registro piu' importante del microprocessore. Diverse istruzioni in Linguaggio Macchina consentono di copiare il contenuto di una locazione di memoria nell'accumulatore, di copiare il contenuto dell'accumulatore in una locazione di memoria, di modificare direttamente il contenuto dell'accumulatore o di qualche altro registro, senza interessare la memoria. E' l'unico registro fornito di istruzioni per eseguire calcoli aritmetici.

REGISTRO INDICE X

Ci sono istruzioni relative a quasi tutte le trasformazioni che si possono fare all'accumulatore. Ma ci sono altre istruzioni per cose che solamente il registro X puo' effettuare. Diverse istruzioni in Linguaggio Macchina permettono di copiare il contenuto di una locazione di memoria del registro X, di copiare il contenuto dal registro X in una locazione di memoria e di modificare il contenuto del registro X o di qualche altro registro direttamente, senza interessare la memoria.

REGISTRO INDICE Y

Ci sono istruzioni per quasi tutte le trasformazioni che si possono effettuare sull'accumulatore e sul registro X. Ma ci sono altre istruzioni che solo il registro Y puo' eseguire. Numerose istruzioni in Linguaggio Macchina consentono di copiare il contenuto di una locazione di memoria nel registro Y, di copiare il contenuto del registro Y in una locazione di memoria, e di modificare il contenuto di Y o di qualche altro registro direttamente, senza interessare altra memoria.

REGISTRO DI STATO

Registro costituito da 8 "flag" (flag = indicatore di un fatto, avvenuto o no).

CONTATORE DI PROGRAMMA (PROGRAM COUNTER)

Contiene l'indirizzo dell'istruzione corrente in Linguaggio Macchina che sta per essere eseguita. Poiche' il Sistema Operativo e' sempre "caricato" (RUNning) sul COMMODORE 64 (o su qualche altro computer con queste caratteristiche), il contatore di programma viene sempre modificato. Puo' essere fermato solamente dall'arresto del microprocessore.

PUNTATORE ALLO STACK (STACK POINTER)

Contiene la locazione del primo posto vuoto sullo "stack" (pila). Lo stack viene usato dai programmi in Linguaggio Macchina e dal computer per memorizzazioni temporanee.

PORTA DI INPUT/OUTPUT

Questo registro appare nella locazioni di memoria 0 (per il registro DATA DIRECTION) e 1 (per la PORTA attuale). Si tratta di una porta di input/output a 8 bit. Sul COMMODORE 64 questo registro e' usato per la gestione della memoria, per permettere al circuito di controllare piu' di 64K di memoria RAM e ROM. I particolari di questi registri verranno esposti piu' avanti.

COME SI SCRIVONO I PROGRAMMI IN LINGUAGGIO MACCHINA?

Poiche' i programmi in Linguaggio Macchina risiedono in memoria, e dato che non e' facile scrivere ed editare programmi in Linguaggio Macchina sul COMMODORE 64, per fare cio' si deve usare un programma particolare, oppure scrivere un programma in BASIC che permetta di scrivere in Linguaggio Macchina.

I piu' comuni metodi usati per scrivere programmi in Linguaggio Macchina sono i programmi assembler. Questi pacchetti ("packages") consentono di scrivere istruzioni in Linguaggio Macchina in un formato mnemonico standardizzato, che rende il programma in Linguaggio Macchina molto piu' leggibile di un flusso di numeri. Riassumendo: un programma che permette di scrivere in formato numerico programmi in Linguaggio Macchina e' chiamato assemblatore. Per inciso, un programma che risultasse scritto in linguaggio macchina in formato mnemonico, e' chiamato disassembler. A disposizione del COMMODORE 64 c'e' una cartuccia (con assembler/disassembler, ecc.), costruita dalla Commodore, per il controllo dei programmi in Linguaggio Macchina:

64MON

La cartuccia 64MON permette di uscire dal mondo del CBM BASIC, per entrare nel Linguaggio Macchina. Essa e' in grado di visualizzare il contenuto dei registri interni del microprocessore 6510, permettendo quindi di visulalizzare parti della memoria e di cambiarle sullo schermo tramite l'Editor di Schermo Incorpora anche l'assembler ed il disassembler, oltre a molte altre caratteristiche che permettono di scrivere e di editare facilmente programmi in Linguaggio Macchina. Per scrivere in Linguaggio Macchina NON E' NECESSARIO usare un assembler, ma con quest'ultimo il compito diventa considerevolmente piu' facile. Se si desidera scrivere programmi in Linguaggio Macchina, e' fortemente consigliabile acquistare un assembler, in assenza del quale si dovra' probabilmente "caricare" (POKE) il programma in Linguaggio Macchina nella memoria, che e' totalmente intrattabile. Da ora in poi, questo manuale riportera' gli esempi nel formato utilizzato dal 64MON. Quasi tutti i formati assembler sono gli stessi, quindi gli esempi in linguaggio macchina riportati sono quasi certamente compatibili con qualsiasi assembler. Ma prima di esporre altre caratteristiche del 64MON, occorre spiegare il sistema di numerazione esadecimale.

NOTAZIONE ESADECIMALE

La notazione esadecimale e' usata dalla maggior parte dei programmatori in linguaggio macchina quando trattano un numero o un

indirizzo in linguaggio macchina.
Alcuni assembler consentono di riferirsi ad indirizzi e **numeri in** decimale (base 10), in binario (base 2), in ottale **(base 8) ed in** esadecimale (base 16 - indicata in questo manuale con **"HEX"). Questi** **assembler** effettuano la conversione da una base **all'altra in modo** automatico.
Probabilmente, l'esadecimale risulta un po' **difficile da capire** all'inizio, ma come la maggior parte delle cose, **con la pratica non ci** vorra' molto tempo per apprenderlo.
Osservando i numeri decimali (base 10), si puo' **vedere che ciascuna** cifra risulta compresa tra zero **ed il numero della base diminuito** di uno (in questo caso, 9). **QUESTO E' VERO PER TUTTE LE BASI NUMERICHE. I** numeri binari (base 2) sono **rappresentati con cifre comprese fra zero** e uno (quest'ultimo valore **si ottiene diminuendo la base 2 di una** unita'). Similmente, i **numeri esadecimali hanno** cifre comprese **fra zero** e quindici, solo che non ci sono singole cifre in grado di rappresentare i numeri **da dieci a quindici**; pertanto, vengono utilizzate le **prime sei cifre dell'alfabeto**:

DECIMALE	ESADECIMALE	BINARIO
0	0	00000000
1	1	00000001
2	2	00000010
3	3	00000011
4	4	00000100
5	5	00000101
6	6	00000110
7	7	00000111
8	8	00001000
9	9	00001001
10	A	00001010
11	B	00001011
12	C	00001100
13	D	00001101
14	E	00001110
15	F	00001111
16	10	00010000

Un esempio di come costruire una base (numero) decimale puo' **essere** il **seguente**:

Base elevata a
potenze decrescenti. . . . 10^3 10^2 10^1 10^0

Equivale a 1000 100 10 1

Esempio: 4569 (base 10) 4 5 6 9

$=(4\times1000)+(5\times100)+(6\times10)+9$

In maniera del tutto **analoga si** puo' **costruire** una **base** (numero) esadecimale:

Base elevata a
potenze decrescenti... $16^3\ 16^2\ 16^1\ 16^0$

Equivale a 4096 256 16 1

Esempio: 11D9 (base 16) 1 1 D 9

$=1\times4096+1\times256+13\times16+9$

Quindi, 4569 (base 10) = 11D9 (base 16).
L'intervallo delle locazioni di memoria indirizzabili (come visto precedentemente) e' 0...65535; in notazione esadecimale, diviene 0...FFFF.
Di solito i numeri esadecimali sono preceduti dal simbolo "$", in modo da poterli distinguere dai numeri decimali. Numeri esadecimali possono essere osservati, usando la cartuccia 64MON, visualizzando il contenuto di una parte di memoria. Digitando:

```
B■
     PC SR AC XR YR SP
   .; 0401 32 04 5E 00 F6
```

(possono comparire anche valori differenti)

.M 0000 0020 (seguito da RETURN).

compaiono file di 6 numeri "HEX". Il primo numero di 4 cifre rappresenta l'indirizzo del primo byte di memoria visualizzato; gli altri cinque numeri costituiscono il contenuto reale della locazione di memoria che inizia da quell'indirizzo.
E' cosigliabile cercare di imparare a "pensare" in esadecimale: non e' poi cosi' difficile, poiche' non e' necessaria la riconversione in decimale; infatti, non fa alcuna differenza dire che un particolare valore e' memorizzato in $14ED anziche' 5357 (corrispondente valore decimale).

LA PRIMA ISTRUZIONE IN LINGUAGGIO MACCHINA

LDA - Carica l'Accumulatore

Nel linguaggio assembly del 6510, i codici mnemonici sono quasi sempre di tre caratteri. LDA sta per "carica l'accumulatore con..."; cio' che deve essere caricato nell'accumulatore e' specificato dal parametro (o dai parametri) associato all'istruzione stessa. L'assembler sa che il "token" e' rappresentato dal corrispondente codice mnemonico, e quando "assembla" un'istruzione trasferisce semplicemente in memoria (a qualunque indirizzo specificato) il "token" ed i parametri associati. Alcuni assembler ritornano messaggi di errore, oppure avvertono quando si cerca di assemblare qualcosa che l'assembler o il microprocessore 6510 non possono fare.
Se al parametro associato all'istruzione si antepone il simbolo "#", cio' significa che si intende caricare il registro specificato nell'istruzione con il "valore" che segue "#". Per esempio:

LDA #$05 ← ($ = HEX)

Questa istruzione mette $05 (5 esadecimale) nel registro accumulatore. Nell'indirizzo specificato per questa istruzione, l'assembler carica $A9 (che in questo caso e' il "token"

dell'istruzione) nell'indirizzo specificato, e $05 nella prima locazione seguente quella contenente l'istruzione stessa ($A9). Se il parametro che deve essere utilizzato e' preceduto da "#", cioe' se il parametro rappresenta un "valore" piuttosto che il contenuto di una locazione di memoria o di un altro registro, allora l'istruzione si dice "immediata". Per illustrare meglio quanto detto, vediamo un altro modo.

Se si vuole trasferire il contenuto della locazione di memoria $102E nell'accumulatore, si deve usare la seguente istruzione "assoluta":

```
LDA $102E
```

L'assembler e' in grado di distinguere tra i due differenti modi in quanto il secondo non ha "#" prima del parametro. Il microprocessore 6510 puo' distinguere tra il modo immediato ed il modo assoluto per il fatto che sono formati da "token" leggermente diversi: LDA (immediato) ha come "token" $09, mentre LDA (assoluto) ha come "token" $AD.

Il codice mnemonico che rappresenta un'istruzione indica in genere l'azione svolta dall'istruzione stessa. Consideriamo ad esempio un'altra istruzione: LDX; che cosa pensiamo che faccia ?

Se si risponde "carica il registro X con...", allora...siamo i primi della classe ! Altrimenti, non c'e' di che preoccuparsi, imparare il linguaggio macchina richiede pazienza, e non puo' essere appreso in un solo giorno.

I vari registri interni possono essere pensati come locazioni di memoria speciali, poiché possono contenere un solo byte. Il sistema di numerazione binario (base 2) non richiede particolari spiegazioni: valgono per esso le stesse regole riportate per i precedenti sistemi esadecimale e decimale; ci limitiamo a ricordare che un "bit" rappresenta una cifra binaria, e che otto bit costituiscono un byte. Cio' significa che il massimo numero che un byte puo' contenere e' dato dal piu' grande numero esprimibile con otto cifre binarie, vale a dire 11111111 (binario), equivalente a $FF (esadecimale) e a 255 (decimale). Probabilmente, ci si potra' meravigliare del fatto che in una locazione di memoria possa essere contenuto un numero compreso solamente fra 0 e 255. Se si prova a digitare POKE 7680,260 (istruzione BASIC che dice di "caricare il numero 260 nella locazione di memoria 7680), allora, dato che l'interprete BASIC sa che una locazione di memoria puo' contenere un numero comreso solamente fra 0 e 255, il COMMODORE 64 replichera':

```
?ILLEGAL QUANTITY ERROR

READY.
■
```

Se il limite di un byte e' $FF (HEX), come viene espresso in memoria il parametro indirizzo contenuto nell'istruzione assoluta "LDA $10E" ? Elementare: con due byte (ovvio, visto che uno non basta...). Le due cifre basse (le piu' a destra) dell'indirizzo esadecimale formano il "byte basso" dell'indirizzo, quelle alte (le piu' a sinistra) il "byte alto".

Il 6510 richiede che ciascun indirizzo venga specificato ponendo per primo il byte basso, seguito da quello alto. Cio' significa che l'istruzione "LDA $102E" e' rappresentata in memoria dai tre valori consecutivi:

$AD, $2E, $10

Per poter scrivere un programma e' necessario conoscere ancora un'istruzione: BRK. Una completa descrizione di questa istruzione si trova in un Manuale di Programmazione del 6502 M.O.S. Per il momento, si puo' pensare a questa istruzione come corrispondente in linguaggio macchina della END del BASIC.
Se si scrive un programma usando la cartuccia 64MON, sistemando l'istruzione BRK alla fine del programma, questa istruzione, una volta eseguito e terminato il programma, fa tornare alla 64MON. Se cio' non accade, vuol dire che si e' verificato un errore nel programma, oppure l'istruzione BRK non e' stata mai raggiunta (proprio come se non si fosse mai raggiunta una END in un programma BASIC). Si capisce cosi' che se il COMMODORE 64 non avesse una chiave di STOP, non saremmo in grado di abortire i programmi BASIC!

IL PRIMO PROGRAMMA

Se si e' usata l'istruzione POKE in un programma BASIC per trasferire i caratteri sullo schermo, allora si e' gia' a conoscenza del fatto che i codici carattere per "modificare tramite POKE" sono diversi dai valori carattere del CBM ASCII. Se, ad esempio, digitiamo:

PRINT ASC("A") (premere poi RETURN)

il COMMODORE 64 risponde con:

```
65

READY.
■
```

Tuttavia, per inserire una "A" sullo schermo tramite l'istruzione POKE, digitare (il codice per "A" e' 1):

SHIFT CLR/HOME per pulire lo schermo

POKE 1024,1 (e RETURN) (1024 e' l'inizio della memoria schermo)

La lettera "P" dell'istruzione POKE dovrebbe essere ora una "A". Proviamo ora in linguaggio macchina: digitiamo quanto segue su 64MON (da questo momento, il cursore dovrebbe essere intermittente a fianco del" "):

.A 1400 LDA #$01 (premere poi RETURN)

Il COMMODORE 64 risponde con:

```
.A 1400 LDA #$01
.A 1402 ■
```

A questo punto digitiamo:

.A 1402 STA $0400

(l'istruzione STA memorizza il contenuto dell'accumulatore in una locazione di memoria specifica). La risposta del COMMODORE 64 e':

```
.A 1405 ■
```

Digitiamo infine:

```
.A 1405 BRK
```

Puliamo lo schermo e digitiamo:

```
G 1400
```

Se tutto e' stato eseguito correttamente, la "G" si trasforma in "A". Si e' scritto cosi' un primo programma in linguaggio macchina, che ha lo scopo di memorizzare un carattere ("A") nella prima locazione della memoria schermo. Possiamo a questo punto introdurre altre istruzioni e principi.

MODI DI INDIRIZZAMENTO

PAGINA ZERO

Come abbiamo gia' visto, gli indirizzi assoluti sono espressi in termini di byte alto e basso. Il byte alto si riferisce alla pagina di memoria. Per esempio, l'indirizzo $1637 si trova nella pagina $16 (22 decimale), mentre l'indirizzo $0277 si trova nella pagina $02 (2 decimale). Esiste, tuttavia, un particolare modo di indirizzamento, conosciuto come "indirizzamento di pagina zero", associato, come spiegato dallo stesso nome, all'indirizzamento delle locazioni di memoria di pagina zero. Questi indirizzi, percio', hanno SEMPRE il byte alto a zero. Questo modo di indirizzamento si aspetta solo un byte per la descrizione dell'indirizzo, invece dei due byte usati per un indirizzo assoluto. Il modo di indirizzamento di pagina zero dice al microprocessore di assumere zero come indirizzo alto, per cui puo' fare riferimento a locazioni di memoria i cui indirizzi vanno da $0000 a $00FF. Questo ora puo' sembrare poco importante, ma presto ci serviranno i principi dell'indirizzamneto di pagina zero.

LO STACK

Il microprocessore 6510 ha uno "stack", che viene usato sia dal programmatore che dal microprocessore per memorizzazioni temporanee, come ad esempio una lista ordinata di DATI. L'istruzione GOSUB del BASIC, che permette al programmatore di richiamare una sottoprocedura, deve tener presente in quale punto del programma sta per essere chiamata, in modo che, quando nella sottoprocedura viene eseguita l'istruzione RETURN, l'interprete BASIC "sappia" da quale punto del programma riprendere l'esecuzione. Quando incontra in un programma l'istruzione GOSUB, l'interprete BASIC, prima di mandare in esecuzione la sottoprocedura, ne carica la posizione attuale sullo "stack"; dopo l'esecuzione dell'istruzione RETURN, l'interprete preleva dallo "stack" l'informazione che gli comunica in quale punto del programma si trovava prima che fosse eseguita la sottoprocedura chiamata. L'interprete fa uso di istruzioni come PHA, che carica sullo "stack" il contenuto dell'accumulatore, e come PLA (opposta di PHA).

che preleva un valore dallo "stack" caricandolo nell'accumulatore. Anche il registro di stato puo' essere caricato o scaricato, facendo uso rispettivamente delle istruzioni PHP e PLP.
Lo stack e' lungo 256 byte ed e' allocato nella pagina 1 della memoria: si estende, quindi, da $0100 a $01FF, e nella memoria e' organizzato in senso inverso. In altre parole, la prima posizione dello "stack" ha indirizzo $01FF, e l'ultima $0100. Un altro registro del microprocessore 6510 e' il PUNTATORE ALLO STACK, che indica sempre la prima posizione disponibile sullo "stack". Quando un dato viene inserito nello stack, trova posto nella locazione indicata dal puntatore allo stack, quindi il puntatore allo stack viene fatto scendere alla prossima locazione (decrementato). Quando invece un dato viene tolto dallo stack, il puntatore allo stack viene incrementato ed il byte puntato da questo puntatore viene sistemato nel registro specificato.
Fino a questo punto sono state esaminate le istruzioni scritte in modo immediato, pagina zero ed assoluto; introduciamo ora il modo "implicito". Il modo implicito indica che l'informazione e' implicita nell'istruzione stessa, cioe' a quali registri, indicatori (flag) e memoria fa riferimento l'istruzione. Gli esempi che abbiamo visto sono PHA, PLA, PHP e PLP, che si riferiscono, rispettivamente, all'elaborazione dello stack (le prime due) ed ai registri di stato e dell'accumulatore (le seconde due).

NOTA: Da ora in avanti, indicheremo con X il registro X, con A l'accumulatore, con Y il registro Y, con S il puntatore allo stack e con P lo stato del processore.

INDICIZZAZIONE

L'indicizzazione occupa una parte importante nell'elaborazione del microprocessore 6510. Puo' essere definita come "la generazione di un indirizzo attuale ottenuto sommando all'indirizzo base il contenuto di entrambi i registri indice X e Y".
Per esempio, se X contiene $05 ed il microprocessore esegue un'istruzione LDA nel "modo indicizzato assoluto X" con un indirizzo base (ad esempio, $9000), allora la locazione reale riportata nel registro A e' data da $9000+$05=$9005. Il formato del codice mnemonico di un'istruzione indicizzata assoluta e' lo stesso di quello di un'istruzione assoluta, ad eccezione di una X o di una Y indicanti che l'indice viene sommato all'indirizzo.

ESEMPIO:

```
LDA $9000,X
```

Disponibili sul microprocessore 6510 ci sono modi di indirizzamento indicizzato assoluto, indicizzato di pagina zero, indicizzato indiretto ed indiretto indicizzato.

INDICIZZATO INDIRETTO

E' l'unico modo che permette l'uso del registro Y come indice. L'indirizzo reale puo' essere solamente nella pagina zero. Il modo dell'istruzione e' chiamato indiretto perche' l'indirizzo di pagina zero, specificato nell'istruzione, contiene il byte basso dell'indirizzo reale, ed il byte successivo a questo il byte alto.

ESEMPIO:

Supponiamo che la locazione $01 contenga $45 e la locazione $02 contenga $1E. Se viene eseguita l'istruzione per caricare l'accumulatore nel modo indicizzato indiretto, e se l'indirizzo di pagina zero specificato e' $01, allora l'indirizzo reale sara':

```
Byte basso = Contenuto di $01
Byte  alto = Contenuto di $02
Registro Y = $00
```

Quindi l'indirizzo reale sara' dato da $1045+Y=$1045.

L'intestazione di questo modo implica in effetti un principio indiretto, anche se a prima vista risulta difficile da afferrare. In un'altra forma, tale principio puo' suonare come segue: "Sto andando a recapitare questa lettera all'ufficio postale di indirizzo $01, MEMORY ST., e l'indirizzo riportato sulla lettera si trova $05 abitazioni oltre $1600, MEMORY ST.". Traducendo in codice:

```
LDA #$00      - Carica l'indirizzo basso di base attuale
STA $02       - Imposta il byte basso dell'indirizzo indiretto
LDA #$16      - Carica l'indirizzo indiretto alto
STA $03       - Imposta il byte alto dell'indirizzo indiretto
LDY #$05      - Imposta l'indice indiretto
LDA ($02),Y   - Carica indirettamente indicizzato da Y
```

INDIRETTO INDICIZZATO

E' l'unico modo che consente l'uso del registro X come indice. Questo modo e' analogo al modo indicizzato indiretto, ad eccezione del fatto che viene indicizzato l'indirizzo di pagina zero del PUNTATORE, anziche' l'indirizzo di base attuale. Percio', l'indirizzo di base attuale E' l'indirizzo attuale, in quanto l'indice e' gia' stato usato in modo indiretto. Il modo indiretto indicizzato puo' essere usato anche nel caso in cui nella memoria di pagina zero venga allocata una TABELLA di puntatori indiretti; in questo caso, il registro X specifica quale puntatore indiretto usare.

ESEMPIO:

Supponiamo che la locazione $02 contenga $45 e la locazione $03 contenga $10. Se si esegue l'istruzione per caricare l'accumulatore nel modo indiretto indicizzato, e se l'indirizzo di pagina zero specificato e' $02, allora l'indirizzo attuale sara':

```
Byte basso = Contenuto di ($02+X)
Byte  alto = Contenuto di ($03+X)
Registro X = $00
```

Percio' il puntatore attuale si trovera' in $02+X=$02, e l'indirizzo attuale sara' l'indirizzo indiretto contenuto in $02, cioe' di nuovo $1045
L'intestazione di questo modo rende in effetti implicito il principio, anche se a prima vista risulta difficile da afferrare. In un'altra forma, tale principio puo' suonare come segue: "Sto andando a recapitare questa lettera al quinto ufficio postale di indirizzo $01, MEMORY ST., e l'indirizzo riportato sulla lettera sara' poi consegnato a $1600, MEMORY ST.". Traducendo in codice:

```
LDA #$00     - Carica l'indirizzo basso di base attuale
STA $06      - Imposta il byte basso dell'indirizzo indiretto
LDA #$16     - Carica l'indirizzo indiretto alto
STA $07      - Imposta il byte alto dell'indirizzo indiretto
LDX #$05     - Imposta l'indice indiretto (X)
LDA ($01,X)  - Carica indicizzato indirettamente da X
```

NOTA: Dei due modi indiretti di indirizzamento, quello di uso piu' comune e' il primo (indicizzato indiretto)

SALTI E CONTROLLO

Un altro principio molto importante nel linguaggio macchina e' dato dalla capacita' di testare e di scoprire certe condizioni, in modo simile alla struttura "IF...THEN, IF...GOTO" del CBM BASIC.
I numerosi indicatori posti nel registro di stato vengono interessati da istruzioni diverse in maniera diversa. Per esempio, c'e' un indicatore che viene impostato (ON) quando un'istruzione da' come risultato zero, e viene disattivato (OFF) quando il risultato e' diverso da zero.
L'istruzione:

```
LDA #$00
```

attiva l'indicatore di risultato zero, in quanto questa istruzione ha come risultato, nell'accumulatore, proprio lo zero.
Esistono alcune istruzioni che in particolari condizioni fanno passare il controllo ad un'altra parte del programma. Un esempio di istruzione di salto e' BEQ, che significa "salta se il risultato e' uguale a zero". Le istruzioni di salto vengono eseguite se la condizione risulta vera, altrimenti il programma continua dalla prossima istruzione, come se non fosse accaduto nulla. Le istruzioni di salto vengono eseguite non in base al risultato della precedente (o delle precedenti) istruzione, ma dall'esame interno del registro di stato. Come si e' gia' detto, nel registro di stato c'e' un indicatore per risultato zero. Quindi, l'istruzione BEQ salta se l'indicatore di risultato zero (conosciuto come Z) e' attivato. Ogni istruzione di salto ha la sua complementare. L'istruzione BEQ ha l'istruzione complementare BNE, che significa "salta se il risultato non e' uguale a zero" (se cioe' Z non e' attivato).
I registri indice hanno un numero di istruzioni associate che ne modificano il contenuto. Per esempio, l'istruzione INX incrementa il registro indice X. Se il registro indice X, prima di essere incrementato, conteneva $FF (il massimo numero che il registro X puo' contenere), "ritornera'" zero. Se si vuole che il programma continui a fare qualcosa fintantoche' si incrementa l'indice X fino a portarlo a zero, si puo' usare l'istruzione BNE, che continua a testare X finche'

non lo trova a zero.
Il contrario dell'istruzione INX e' DEX, che decrementa il registro indice X. Se il contenuto del registro X e' zero, DEX lo riporta a $FF. In maniera del tutto analoga si comportano le istruzioni INY e DEY, relative al registro Y.
Ma che cosa fare se un programma non desidera aspettare che X e Y abbiano raggiunto (oppure no) lo zero ?. Bene, ci sono le istruzioni di confronto, CPX e CPY, che consentono (al programmatore che lavora in Linguaggio Macchina) di testare il registro indice con valori specifici, o addirittura con il contenuto delle locazioni di memoria. Se si volesse sapere se il registro X contiene $40, si dovrebbe usare la seguente istruzione:

```
CPX #$40      - Confronta X con il "valore" $40
BEQ           - Se questa condizione e' vera, salta ad un'altra
                parte del programma
```

Il confronto e le istruzioni di salto coprono la parte piu' importante di ogni Linguaggio Macchina.
L'operando specificato in un'istruzione di salto, usando la 64MON, rappresenta l'indirizzo della parte di programma alla quale si salta quando si incontrano le dovute condizioni. In ogni caso, l'operando e' solamente un "offset", che conduce il programma dal punto in cui si trova attualmente all'indirizzo specificato. Tale "offset" e' di appena un byte, e percio' l'intervallo a cui un istruzione di salto puo' saltare e' limitato: infatti, sono consentiti salti di 128 byte all'indietro e di 127 in avanti.

NOTA: Questo e' un intervallo totale di 255 byte, che rappresenta, naturalmente, il massimo intervallo di valori che un byte puo' contenere.

La cartuccia 64MON segnala un "salto fuori dall'intervallo", rifiutandosi di "assemblare" quella particolare istruzione. Il salto e' un'istruzione "veloce" data dagli standard del Linguaggio Macchina, grazie al principio dell'"offset", che e' l'opposto di un indirizzo assoluto. La 64MON permette di scrivere un indirizzo assoluto, ed e' essa a calcolare l'esatto "offset". Questa e' una delle "comodita'" date dall'uso di un assembler.

NOTA: NON e' possibile trattare ogni singola istruzione di salto. Per ulteriori informazioni si rimanda alla sezione bibliografica (Appendice F)

SOTTOPROCEDURE

Nel Linguaggio Macchina (analogamente a quanto avviene nel BASIC) e' possible chiamare una sottoprocedura. L'istruzione che chiama una sottoprocedura e' JSR (salta alla procedura), seguita da uno specifico indirizzo assoluto.
C'e' una sottoprocedura, incorporata nel Sistema Operativo, che scrive un carattere sullo schermo. Il codice CBM ASCII dei caratteri deve trovarsi nell'accumulatore, prima della chiamata alla sottoprocedura, il cui indirizzo e' $FFD2; percio', per scrivere "HI" sul video, si deve caricare il seguente programma:

```
.A 1400 LDA #$48    - Carica il codice CBM ASCII della lettera "H"
.A 1402 JSR $FFD2   - Stampa tale lettera
.A 1405 LDA #$49    - Carica il codice CBM ASCII della lettera "I"
.A 1407 JSR $FFD2   - Stampa tale lettera
.A 140A LDA #$0D    - Stampa un ritorno carrello
.A 140C JSR $FFD2
.A 140F BRK         - Ritorna a 64MON
.G 1400             - Stampa "HI" e ritorna a 64MON
```

La procedura "stampa di un carattere", che abbiamo appena usato, fa parte della tavola di salto del KERNAL. L'istruzione equivalente alla GOTO del BASIC e' JMP, che significa "salto all'indirizzo assoluto specificato". Il KERNAL e' una lunga lista di sottoprocedure "standardizzate", che controllano tutto l'input e l'output del COMMODORE 64. Ciascuna entrata nel KERNAL salta ad una sottoprocedura del Sistema Operativo. Questa "tavola di salto" si trova fra le locazioni di memoria $FF84 e $FFF5. Una completa spiegazione del KERNAL e' disponibile nella "sezione di riferimento al KERNAL" di questo Manuale. Tuttavia, certe procedure vengono usate qui di seguito per mostrare quanto sia efficace e di facile uso il KERNAL.
Usiamo ora i nuovi principi appresi in un altro programma, che ci consente di inserire le istruzioni nel contesto.
Questo programma visualizza l'alfabeto usando una routine del KERNAL. L'unica nuova istruzione introdotta in questo programma e' TXA, che trasferisce il contenuto del registro indice X nell'accumulatore.

```
.A 1400 LDX #$41    - X = CBM ASCII di "A"
.A 1402 TXA         - A = X
.A 1403 JSR $FFD2   - Scrive il carattere
.A 1406 INX         - Incrementa il contatore
.A 1407 CPX #$5B    - Si e' oltrepassata la "Z" ?
.A 1409 BNE $1402   - Se no, torna all'istruzione 1402 e ricomincia
.A 140B BRK         - Se si, ritorna a 64MON
```

Per far stampare l'alfabeto al COMMODORE 64, usare il solito comando:

```
.G 1400
```

Il commento che si trova a lato del programma ne spiega la logica ed il funzionamento. E' sempre consigliabile scrivere un programma prima sulla carta, e quindi verificarlo, possibilmente, un segmento alla volta.

SUGGERIMENTI UTILI PER IL PRINCIPIANTE

Uno dei modi migliori per imparare il Linguaggio Macchina e' osservare programmi scritti in Linguaggio Macchina da altre persone, pubblicati di solito sulle riviste e sugli articoli della stampa specializzata. Non ha alcuna importanza se il computer per cui sono scritti e' diverso dal COMMODORE 64, basta che anch'esso usi il microprocessore 6510 (o 6502). Si dovrebbe essere in grado di comprendere completamente i codici che vi sono riportati. Fare questo richiede perseveranza, specialmente quando si incontra una tecnica nuova mai vista prima.
Dopo aver osservato attentamente gli altri programmi in Linguaggio Macchina, se ne dovrebbero scrivere alcuni, come ad esempio programmi

di utilita' per altri programmi in BASIC, oppure programmi scritti interamente in Linguaggio Macchina.
Inoltre, si dovrebbero usare le "utilities", disponibili o NEL computer o in un programma, che saranno di aiuto nella scrittura, nell'editazione o nell'individuazione degli errori di un programma in Linguaggio Macchina. Un esempio puo' essere fornito dal KERNAL, che permette di controllare la tastiera, di stampare testi, di controllare i dispositivi periferici come dischi, stampanti, modem, ecc., la gestione della memoria e dello schermo. E' estremamente potente, ed il suo uso e' fortemente raccomandato (cfr. Gestione del KERNAL).
Vantaggi derivanti dallo scrivere i programmi in Linguaggio Macchina:

1. Velocita' - Il Linguaggio Macchina e' centinaia, talvolta migliaia di volte, piu' veloce di un linguaggio ad alto livello come il BASIC.

2. Tenuta - Un Linguaggio Macchina puo' essere reso completamente impermeabile, cioe' si puo' mettere l'Utente nella condizione di eseguire solo quello che gli consente di fare il programma. In un linguaggio ad alto livello si puo' solo sperare che l'Utente non faccia "saltare" l'interprete BASIC, inserendo ad esempio uno zero che, successivamente, comporti un:

```
?DIVISION BY ZERO ERROR IN LINE 830

READY.
■
```

In essenza, la programmazione in Linguaggio Macchina non puo' che valorizzare maggiormente il computer.

UN COMPITO PIÙ IMPEGNATIVO

Affrontando un compito piu' impegnativo, si e' di solito presi da pensieri. Si cerca di intuire come vengono eseguiti certi processi in Linguaggio Macchina. Quando si e' all'inizio di un compito, e' buona norma riportarlo su carta. E' consigliabile inoltre fare uso di diagrammi a blocchi di uso della memoria, di moduli funzionali di codici e di flussi di programma. Supponiamo di voler scrivere il gioco della roulette in Linguaggio Macchina; ad esempio:

* Visualizzare il titolo
* Chiedere se il giocatore richiede istruzioni
* Se si, visualizzarle e saltare a START
* Se no, saltare direttamente a START
* START - Inizializzare tutto
* MAIN - Visualizzare il tavolo della roulette
* Raccogliere le puntate
* Far girare la ruota
* Far rallentare la ruota fino a fermarla
* Confrontare le puntate con il risultato
* Informare il giocatore
* Ha perso ?
* Se si, saltare a MAIN
* Se no, informarlo della vincita e saltare a START

Questo e' lo schema principale. Quando ciascun modulo e' stato individuato, puo' essere scomposto ulteriormente. Se si deve affrontare un problema complesso che puo' essere scomposto in blocchi piu' piccoli per poter essere compreso, allora siamo in grado di affrontare qualcosa che sembra impossibile, e vederlo realizzato.
Questo processo migliora solo con la pratica, percio' TENETE DURO!...

INSIEME DELLE ISTRUZIONI DEL MICROPROCESSORE MCS6510 SEQUENZA ALFABETICA

ADC	Somma la Memoria all'Accumulatore, con Riporto
AND	Fra Memoria ed Accumulatore esegue un "AND" logico
ASL	Scorrimento (Shift) a Sinista di un bit (Memoria o Accumulatore)
BCC	Salto sull'azzeramento del Riporto
BCS	Salto sull'impostazione del Riporto
BEQ	Salto se Risultato Zero
BIT	Confronta i bit nella Memoria con l'Accumulatore
BM1	Salto se Risultato Meno
BNE	Salto se Risultato Non-Zero
BPL	Salto su Risultato Piu'
BRK	Interruzione (break) forzata
BVC	Salto sull'Azzeramento dell'Overflow
BVS	Salto sull'Impostazione dell'Overflow
CLC	Azzera il carry
CLD	Azzera il Modo Decimale
CLI	Azzera l'Iterruzione e Diasabilita il Bit di Interrupt
CLV	Azzera l'Indicatore di Overflow
CMP	Compara Memoria ed Accumulatore
CPX	Compara Memoria ed Indice X
CPY	Compara Memoria ed Indice Y
DEC	Decrementa la Memoria di uno
DEX	Decrementa l'Indice X di uno
DEY	Decrementa l'Indice Y di uno
EOR	OR esclusivo della Memoria con l'Accumulatore
INC	Incrementa la Memoria di uno
INX	Incrementa l'Indice X di uno
INY	Incrementa l'Indice Y di uno
JMP	Salto a Nuova Locazione
JSR	Salto a Nuova Locazione e salvataggio dell'indirizzo di ritorno
LDA	Carica l'Accumulatore con la Memoria
LDX	Carica l'Indice X con la Memoria
LDY	Carica l'Indice Y con la Memoria
LSR	Scorrimento a Destra (Shift) di un Bit (Memoria o Accumulatore)
NOP	Nessuna Operazione

ORA	OR della Memoria con l'Accumulatore
PHA	Salva l'Accumulatore allo Stack
PHP	Posiziona lo Stato del Processore sullo Stack salva il R di Stato
PLA	Carica l'accumulatore con l'ultimo valore dello Stack
PLP	Ritira lo Stato del Processore dallo Stack
ROL	Ruota a Sinistra di un Bit (Memoria o Accumulatore)
ROR	Ruota a Destra di un Bit (Memoria o Accumulatore)
RTI	Ritorno da Interruzione
RTS	Ritorno da Sottoprocedura
SBC	Sottrae la Memoria dall'Accumulatore, con Prestito
SEC	Imposta l'Indicatore di Riporto
SED	Imposta il Modo Decimale
SEI	Disabilita il flag di interrupt dell'interruzione
STA	Registra l'Accumulatore in Memoria
STX	Registra l'Indice X in Memoria
STY	Registra l'Indice Y in Memoria
TAX	Trasferisce l'Accumulatore all'Indice X
TAY	Trasferisce l'Accumulatore all'Indice Y
TSX	Trasferisce il Puntatore allo Stack all'Indice X
TXA	Trasferisce l'Indice X all'Accumulatore
TXS	Trasferisce l'Indice X al Registro dello Stack
TYA	Trasferisce l'Indice Y all'Accumulatore

Al riassunto che segue vanno applicate le seguenti notazioni:

A	Accumulatore
X,Y	Registri indice
M	Memoria
P	Registro di Stato del Processore
S	Puntatore allo Stack
	Cambio
	Nessun cambio
+	Addizione
	AND logico
-	Sottrazione
	OR esclusivo logico
	Trasferimento dallo Stack
	Trasferimento allo Stack
	Trasferimento a
	Trasferimento da
V	OR logico
PC	Contatore di Programma
PCH	Contatore di Programma Alto
PCL	Contatore di Programma Basso
OPER	Operando
#	Modo di indirizzamento immediato

NOTA In cima a ciascuna tavola e' riportato fra parentesi un numero di riferimento (Ref: xx) che rimanda l'Utente alla relativa sezione nel Manuale di Programmazione, in cui l'istruzione e' definita e discussa

ADC Somma la memoria all'accumulatore, con riporto ADC

Operazione: A+M+C → A, C

N Z C I D V

(Ref.: 2.2.1)

✓ ✓ ✓ ≃ — ✓

Modo di indirizzamento	Forma in linguaggio Assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	ADC #Oper	69	2	2
Pagina Zero	ADC Oper	65	2	3
Pagina Zero, X	ADC Oper, X	75	2	4
Assoluto	ADC Oper	6D	3	4
Assoluto, X	ADC Oper, X	7D	3	4 (*)
Assoluto, Y	ADC Oper, Y	79	3	4 (*)
(Indiretto, X)	ADC (Oper, X)	61	2	6
(Indiretto), Y	ADC (Oper), Y	71	2	5 (*)

(*) Aggiunge 1 se si e' oltrepassato il limite della pagina di memoria

AND "AND" logico tra memoria ed accumulatore AND

Operazione: A∧M → A

N Z C I D V

(Ref.: 2.2.3.0)

✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	AND #Oper	29	2	2
Pagina Zero	AND Oper	25	2	3
Pagina Zero, X	AND Oper, X	35	2	4
Assoluto	AND Oper	2D	3	4
Assoluto, X	AND Oper, X	3D	3	4 (*)
Assoluto, Y	AND Oper, Y	39	3	4 (*)
Indiretto, X	AND (Oper, X)	21	2	6
Indiretto, Y	AND (Oper), Y	31	2	5

(*) Aggiunge 1 se si e' oltrepassao il limite della pagina di memoria

ASL Scorrimento a sinistra di un bit ASL

Operazione: C ← [7|6|5|4|3|2|1|ø] ← ø

N Z C I D V

(Ref.: 10.2)

✓ ✓ ✓ — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ASL A	øA	1	2
Pagina Zero	ASL Oper	ø6	2	5
Pagina Zero, X	ASL Oper, X	16	2	6
Assoluto	ASL Oper	øE	3	6
Assoluto, X	ASL Oper, X	1E	3	7

BCC Salto sull'azzeramento del riporto BCC

Operazione: Salto su C=∅ N Z C I D V

(Ref.: 4.1.1.3) — — — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BCC Oper	9∅	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BCS Salto sull'impostazione del riporto BCS

Operazione: Salto su C=1 N Z C I D V

(Ref.:4.1.1.4) — — — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Relativo	BCS Oper	B∅	2	2 (*)

(*) Aggiuge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BEQ Salto su risultato zero BEQ

Operazione: Salto su Z=1 N Z C I D V

(Ref.: 4.1.1.5) — — — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BEQ Oper	F∅	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BIT Confronta i bit nella memoria con l'accumulatore BIT

Operazione: A ∧ M, M7 → N, M6 → V

N Z C I D V

I bit 6 e 7 sono trasferiti nel registro di stato. Se il risultato di A ∧ M e' 0, allora Z=1, altrimenti Z=∅

M_7 √ — — — M_6

(Ref.: 4.2.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	BIT Oper	24	2	3
Assoluto	BIT Oper	2C	3	4

BMI Salto su risultato meno BMI

Operazione: Salto su N=1

N Z C I D V

— — — — — —

(Ref.: 4.1.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BMI Oper	3∅	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BNE Salto su risultato non-zero BNE

Operazione: Salto su Z=0

N Z C I D V

— — — — — —

(Ref.: 4.1.1.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BNE Oper	D∅	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BPL Salto su risultato piu' BPL

Operazione: Salto su N=Ø

N Z C I D V
— — — — — —

(Ref.: 4.1.1.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BPL Oper	1Ø	2	2 (*)

(*) Aggiunge 1 se si e' oltrepassato il limite della pagina di memoria

BRK Interruzione forzata BRK

Operazione: Interruzione forzata PC+2↓P↓

N Z C I D V
— — — 1 — —

(Ref.: 9.11)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	BRK	ØØ	1	7

1. Un comando BRK non puo' essere forzato da un'impostazione di I.

BVC Salto sull'azzeramento dell'overflow BVC

Operazione: Salto su V=0

N Z C I D V
— — — — — —

(Ref.: 4.1.1.8)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Relativo	BVC Oper	5Ø	2	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

BVS Salto sull'impostazione dell'overflow BVS

Operazione: Salto su V=1

N Z C I D V
— — — — — —

(Ref.:4.1.1.7)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Relativo	BVS Oper	7Ø	2	2 (*)

(*) Aggiuge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

CLC — Azzera l'indicatore di riporto — CLC

Operazione: ∅ → C

N Z C I D V
— — ∅ — — —

(Ref.: 3.0.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLC	18	1	2 (*)

(*) Aggiunge 1 se il salto avviene nella stessa pagina di memoria
Aggiunge 2 se il salto avviene in pagine di memoria diverse

CLD — Azzera il modo decimale — CLD

Operazione: ∅ → D

N Z C I D V
— — — — ∅ —

(Ref.: 3.3.2.)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLD	D8	1	2

CLI — Azzera l'interruttore e disabilita il bit — CLI

Operazione: ∅ → I

N Z C I D V
— — — ∅ — —

(Ref.: 3.2.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLI	58	1	2 (*)

CLV — Azzera l'indicatore di overflow — CLV

Operazione: ∅ → V

N Z C I D V
— — — — — ∅

(Ref.: 3.6.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	CLV	B8	1	2

CMP Compara memoria ed accumulatore CMP

Operazione: A-M

N Z C I D V

(Ref.: 4.2.1) ✓ ✓ ✓ - - -

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CMP #Oper	C9	2	2
Pagina Zero	CMP Oper	C5	2	3
Pagina Zero, X	CMP Oper,X	D5	2	4
Assoluto	CMP Oper	CD	3	4
Assoluto, X	CMP Oper, X	DD	3	4 (*)
Assoluto, Y	CMP Oper, Y	D9	3	4 (*)
(Indiretto, X)	CMP (Oper, X)	C1	2	6
(Indiretto), Y	CMP (Oper), Y	D1	2	5

(*) Aggiunge 1 se viene superato il limite delle pagine di memoria

CPX Compara memoria ed indice X CPX

Operazione: X-M

N Z C I D V

(Ref.: 7.8) ✓ ✓ ✓ - - -

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CPX #Oper	E∅	2	2
Pagina Zero	CPX Oper	E4	2	3
Assoluto	CPX Oper	EC	3	4

CPY Compara memoria ed indice Y CPY

Operazione: Y-M

N Z C I D V

(Ref.: 7.9) ✓ ✓ ✓ - - -

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	CPY #Oper	C∅	2	2
Pagina Zero	CPY Oper	C4	2	3
Assoluto	CPY Oper	CC	3	4

DEC Decrementa la memoria di uno DEC

Operazione: M-1→M (Ref.: 10.7)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	DEC Oper	C6	2	5
Pagina Zero, X	DEC Oper, X	D6	2	6
Assoluto	DEC Oper	CE	3	6
Assoluto, X	DEC Oper, X	DE	3	7

DEX Decrementa l'indice X di uno DEX

Operazione: X-1→X (Ref.: 7.6)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	DEX	CA	1	2

DEY Decrementa l'indice Y di uno DEY

Operazione: Y-1→Y (Ref.: 7.7)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	DEY	88	1	2

EOR OR esclusivo fra memoria ed accumulatore EOR

Operazione: A⊻M→A (Ref.: 2.2.3.2)

N Z C I D V
✓ ✓ — — — —

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	EOR #Oper	49	2	2
Pagina Zero	EOR Oper	45	2	3
Pagina Zero, X	EOR Oper, X	55	2	4
Assoluto	EOR Oper	4D	3	4
Assoluto, X	EOR Oper, X	5D	3	4 (*)
Assoluto, Y	EOR Oper, Y	59	3	4 (*)
(Indiretto, X)	EOR (Oper, X)	41	2	6
(Indiretto), Y	EOR (oper), Y	51	2	5

(*) Aggiungere 1 se si supera il limite della pagina di memoria.

INC Incrementa la memoria di uno INC

Operazione: M+1 → M

N Z C I D V

√ √ - - - -

(Ref.: 10.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	INC Oper	E6	2	5
Pagina Zero, X	INC Oper, X	F6	2	6
Assoluto	INC Oper	EE	3	6
Assoluto, X	INC Oper, X	FE	3	7

INX Incrementa l'indice X di uno INX

Operazione: X+1 → X

N Z C I D V

√ √ - - - -

(Ref.: 7.4)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	INX	E8	1	2

INY Incrementa l'indice Y di uno INY

Operazione: Y+1 → Y

N Z C I D V

√ √ - - - -

(Ref.: 7.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	INY	C8	1	2

JMP Salto a nuova locazione JMP

Operazione: (PC+1) → PCL (Ref.: 4.0.2)
(PC+2) → PCH (Ref.: 9.8.1)

N Z C I D V

\- - - - - -

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Assoluto	JMP Oper	4C	3	3
Indiretto	JMP (Oper)	6C	3	5

JSR Salto a nuova locazione e salvataggio dell'indirizzo di ritorno JSR

Operazione: PC+2↓, (PC+1) → PCL
(PC+2) → PCH

N Z C I D V

\- - - - - -

(Ref.: 8.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTES	NUM. CICLI
Assoluto	JSR Oper	20	3	6

LDA Carica l'accumulatore con il contenuto della memoria LDA

Operazione: M → A

N Z C I D V

√ √ - - - -

(Ref.: 2.1.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDA #Oper	A9	2	2
Pagina Zero	LDA Oper	A5	2	3
Pagina Zero, X	LDA Oper, X	B5	2	4
Assoluto	LDA Oper	AD	3	4
Assoluto, X	LDA Oper, X	BD	3	4 (*)
Assoluto, Y	LDA Oper, Y	B9	3	4 (*)
(Indiretto, X)	LDA (Oper, X)	A1	2	6
(Indiretto), Y	LDA (Oper), Y	B1	2	5 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LDX Carica l'indice X con la memoria LDX

Operazione: M → X

N Z C I D V

√ √ - - - -

(Ref.: 7.0)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDX #Oper	A2	2	2
Pagina Zero	LDX Oper	A6	2	3
Pagina Zero, Y	LDX Oper, Y	B6	2	4
Assoluto	LDX Oper	AE	3	4
Assoluto, Y	LDX Oper, Y	BE	3	4 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LDY Carica l'indice Y con la memoria LDY

Operazione: M → Y

N Z C I D V
✓ ✓ - - - -

(Ref.: 7.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	LDY #Oper	A0	2	2
Pagina Zero	LDY Oper	A4	2	3
Pagina Zero, X	LDY Oper, X	B4	2	4
Assoluto	LDY Oper	AC	3	4
Assoluto, X	LDY Oper, X	BC	3	4 (*)

(*) Aggiunge 1 se si supera il limite della pagina di memoria

LSR Scorrimento a destra di un bit (memoria o accumulatore LSR

Operazione: ∅ → | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | → C

N Z C I D V
∅ ✓ ✓ - - -

(Ref.: 10.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	LSR A	4A	1	2
Pagina Zero	LSR Oper	46	2	5
Pagina Zero, X	LSR Oper, X	56	2	6
Assoluto	LSR Oper	4E	3	6
Assoluto, X	LSR Oper, X	5E	3	7

NOP Nessuna operazione NOP

Operazione: Nessuna Operazione (2 cicli)

N Z C I D V
_ _ _ _ _ _

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	NOP	EA	1	2

ORA — OR della memoria con l'accumulatore — ORA

Operazione: A V M → A

NZCIDV: ✓ ✓ — — — —

(Ref.: 2.2.3.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	ORA #Oper	Ø9	2	2
Pagina Zero	ORA Oper	Ø5	2	3
Pagina Zero, X	ORA Oper, X	15	2	4
Assoluto	ORA Oper	ØD	3	4
Assoluto, X	ORA Oper, X	1D	3	4 (*)
Assoluto, Y	ORA Oper, Y	19	3	4 (*)
(Indiretto, X)	ORA (Oper, X)	Ø1	2	6
(Indiretto), Y	ORA (Oper), Y	11	2	5

(*) Aggiunge 1 se si supera la pagina di memoria

PHA — Posiziona l'accumulatore sullo stack — PHA

Operazione: A↓

NZCIDV: — — — — — —

(Ref.: 8.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PHA	48	1	3

PHP — Posiziona lo stato del processore sullo stack — PHP

Operazione: P↓

NZCIDV: — — — — — —

(Ref.: 8.5)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PHP	Ø8	1	3

PLA — Ritira l'accumulatore dallo stack — PLA

Operazione: A↑

NZCIDV: ✓ ✓ — — — —

(Ref.: 8.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PLA	68	1	4

PLP Ritira lo stato del processore dallo stack PLP

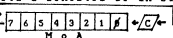
Operazione: P↑

NZCIDV
dallo stack

(Ref.: 8.12)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	PLP	8	1	4

ROL Ruota a sinistra di un bit (memoria o accumulatore) ROL

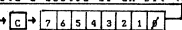
Operazione: ← 7 6 5 4 3 2 1 0 ← C ←
M o A

NZCIDV

(Ref.: 10.3)

✓✓✓---

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ROL A	2A	1	2
Pagina Zero	ROL Oper	26	2	5
Pagina Zero, X	ROL Oper, X	36	2	6
Assoluto	ROL Oper	2E	3	6
Assoluto, X	ROL Oper, X	3E	3	7

ROR Ruota a destra di un bit (memoria o accumulatore) ROR

Operazione: → C → 7 6 5 4 3 2 1 0 →

NZCIDV

(Ref.: 10.4)

✓✓✓---

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Accumulatore	ROR A	6A	1	2
Pagina Zero	ROR Oper	66	2	5
Pagina Zero, X	ROR Oper, X	76	2	6
Assoluto	ROR Oper	6E	3	6
Assoluto, X	ROR Oper, X	7E	3	7

RTI Ritorno da un'interruzione RTI

Operazione: P↑ PC↑

N Z C I D V

(Ref.: 9.6)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	RTI	40	1	6

RTS Ritorno da una subroutine RTS

Operazione: PC↑, PC+1 → PC

N Z C I D V

— — — — — —

(Ref.: 8.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	RTS	60	1	6

SBC Sottrae la memoria dall'accumulatore, con prestito SBC

Operazione: A-M-C → A

N Z C I D V

✓ ✓ ✓ — — ✓

(Ref.: 2.2.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Immediato	SBC #Oper	E9	2	2
Pagina Zero	SBC Oper	E5	2	3
Pagina Zero, X	SBC Oper, X	F5	2	4
Assoluto	SBC Oper	ED	3	4
Assoluto, X	SBC Oper, X	FD	3	4 (*)
Assoluto, Y	SBC Oper, Y	F9	3	4 (*)
(Indiretto, X)	SBC (Oper, X)	E1	2	6
(Indiretto), Y	SBC (Oper), Y	F1	2	5

(*) Aggiunge 1 se si supera il limite della pagina di memoria

SEC Imposta l'indicatore di riporto SEC

Operazione: 1 → C

N Z C I D V

— — 1 — — —

(Ref.: 3.0.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM CICLI
Implicito	SEC	38	1	2

SED Imposta il modo decimale SED

Operazione: 1 → D

N Z C I D V

— — — — 1 —

(Ref.: 3.3.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	SED	F8	1	2

SEI Imposta lo stato di disabilitazione dell'interruzione SEI

Operazione: 1 → I

N Z C I D V

— — — 1 — —

(Ref.: 3.2.1)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	SEI	78	1	2

STA Registra l'accumulatore in memoria STA

Operazione: A → M

N Z C I D V

— — — — — —

(Ref.: 2.1.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STA Oper	85	2	3
Pagina Zero, X	STA Oper, X	95	2	4
Assoluto	STA Oper	8D	3	4
Assoluto, X	STA Oper, X	9D	3	5
Assoluto, Y	STA Oper	99	3	5
(Indiretto, X)	STA (Oper, X)	81	2	6
(Indiretto), Y	STA (Oper), Y	91	2	6

STX Registra l'indice X in memoria STX

Operazione: X → M

N Z C I D V

— — — — — —

(Ref.: 7.2)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STX Oper	86	2	3
Pagina Zero, Y	STX Oper, X	96	2	4
Assoluto	STX Oper	8E	3	4

STY Registra l'indice Y in memoria STY

Operazione: Y → M

N Z C I D V

— — — — — —

(Ref.: 7.3)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Pagina Zero	STY Oper	84	2	3
Pagina Zero, X	STY Oper, X	94	2	4
Assoluto	STY Oper	8C	3	4

TAX Trasferisce l'accumulatore all'indice X TAX

Operazione: A → X

N Z C I D V

✓ ✓ — — — —

(Ref.: 7.11)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TAX	AA	1	2

TAY Trasferisce l'accumulatore all'indice Y TAY

Operaione: A → Y

N Z C I D V

✓ ✓ — — — —

(Ref.: 7.13)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TAY	A8	1	2

TSX Trasferisce il puntatore allo stack nell'indice X TSX

Operazione: S → X

N Z C I D V

✓ ✓ — — — —

(Ref.: 8.9)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TSX	BA	1	2

TXA Trasferisce l'indice X all'accumulatore TXA

Operazione: X → A

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.12)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TXA	8A	1	2

TXS Trasferisce l'indice X al registro dello stack TXS

Operazione: X → S

N Z C I D V
— — — — — —

(Ref.:8.8)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TXS	9A	1	2

TYA Trasferisce l'indice Y all'accumulatore TYA

Operazione: Y → A

N Z C I D V
✓ ✓ — — — —

(Ref.: 7.14)

Modo di indirizzamento	Forma in linguaggio assembler	CODICE OPERATIVO	NUM. BYTE	NUM. CICLI
Implicito	TYA	98	1	2

MODI DI INDIRIZZAMENTO DELLE ISTRUZIONI E RELATIVI TEMPI DI ESECUZIONE (ESPRESSI IN CICLI DI CLOCK)

ACC = Accumulatore
IMM = Immediato
PZ = Pagina Zero
PZX = Pagina Zero, X
PZY = Pagina Zero, Y
ASS = Assoluto
ASX = Assoluto, X
ASY = Assoluto, Y
IMP = Implicito
REL = Relativo
INX = Indiretto, X
INY = Indiretto, Y
IA = Indiretto Assoluto

	ACC	IMM	PZ	PZX	PZY	ASS	ASX	ASY	IMP	REL	INX	INY	IA
ADC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
AND	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
ASL	2	.	5	6	6	7	.	.	.	.	.	.	.
BCC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BCS	.	.	.	.	.	.	.	.	.	2**	.	.	.
BEQ	.	.	.	.	.	.	.	.	.	2**	.	.	.
BIT	.	.	3	.	.	4	.	.	.	.	.	.	.
BMI	.	.	.	.	.	.	.	.	.	2**	.	.	.
BNE	.	.	.	.	.	.	.	.	.	2**	.	.	.
BPL	.	.	.	.	.	.	.	.	.	2**	.	.	.
BRK	.	.	.	.	.	.	.	.	.	.	.	.	.
BVC	.	.	.	.	.	.	.	.	.	2**	.	.	.
BVS	.	.	.	.	.	.	.	.	.	2**	.	.	.
CLC	.	.	.	.	.	.	.	.	2	.	.	.	.
CLD	.	.	.	.	.	.	.	.	2	.	.	.	.
CLI	.	.	.	.	.	.	.	.	2	.	.	.	.
CLV	.	.	.	.	.	.	.	.	2	.	.	.	.
CMP	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
CPX	.	2	3	.	.	4	.	.	.	.	.	.	.
CPY	.	2	3	.	.	4	.	.	.	.	.	.	.
DEC	.	.	5	6	.	6	7	.	.	.	.	.	.
DEX	.	.	.	.	.	.	.	.	2	.	.	.	.
DEY	.	.	.	.	.	.	.	.	2	.	.	.	.
EOR	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
INC	.	.	5	6	.	6	7	.	.	.	.	.	.
INX	.	.	.	.	.	.	.	.	2	.	.	.	.
INY	.	.	.	.	.	.	.	.	2	.	.	.	.
JMP	.	.	.	.	.	3	.	.	.	.	.	.	5

	ACC	IMM	PZ	PZX	PZY	ASS	ASX	ASY	IMP	REL	INX	INY	IA
JSR	.	.	.	.	.	6	.	.	.	.	.	.	.
LDA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
LDX	.	2	3	.	4	4	.	4*	.	.	.	.	.
LDY	.	2	3	4	.	4	4*	.	.	.	.	.	.
LSR	2	.	5	6	.	6	7	.	.	.	.	.	.
NOP	.	.	.	.	.	.	.	.	2	.	.	.	.
ORA	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
PHA	.	.	.	.	.	.	.	.	3	.	.	.	.
PHP	.	.	.	.	.	.	.	.	3	.	.	.	.
PLA	.	.	.	.	.	.	.	.	4	.	.	.	.
PLP	.	.	.	.	.	.	.	.	4	.	.	.	.
ROL	2	.	5	6	.	6	7	.	.	.	.	.	.
ROR	2	.	5	6	.	6	7	.	.	.	.	.	.
RTI	.	.	.	.	.	.	.	.	6	.	.	.	.
RTS	.	.	.	.	.	.	.	.	6	.	.	.	.
SBC	.	2	3	4	.	4	4*	4*	.	.	6	5*	.
SEC	.	.	.	.	.	.	.	.	2	.	.	.	.
SED	.	.	.	.	.	.	.	.	2	.	.	.	.
SEI	.	.	.	.	.	.	.	.	2	.	.	.	.
STA	.	.	3	4	.	4	5	5	.	.	6	6	.
STX	.	.	3	.	4	4	.	.	.	.	.	.	.
STY	.	.	3	4	.	4	.	.	.	.	.	.	.
TAX	.	.	.	.	.	.	.	.	2	.	.	.	.
TAY	.	.	.	.	.	.	.	.	2	.	.	.	.
TSX	.	.	.	.	.	.	.	.	2	.	.	.	.
TXA	.	.	.	.	.	.	.	.	2	.	.	.	.
TXS	.	.	.	.	.	.	.	.	2	.	.	.	.
TYA	.	.	.	.	.	.	.	.	2	.	.	.	.

* Aggiungere un ciclo se l'indicizzazione supera il limite della pagina di memoria

** Aggiungere un ciclo se si effettua un salto, piu' un ciclo se l'operazione di salto supera il limite della pagina di memoria

ØØ - BRK
Ø1 - ORA - (Indirect,X)
Ø2 - Future Expansion
Ø3 - Future Expansion
Ø4 - Future Expansion
Ø5 - ORA - Zero Page
Ø6 - ASL - Zero Page
Ø7 - Future Expansion
Ø8 - PHP
Ø9 - ORA - Immediate
ØA - ASL - Accumulator
ØB - Future Expansion
ØC - Future Expansion
ØD - ORA - Absolute
ØE - ASL - Absolute
ØF - Future Expansion
1Ø - BPL
11 - ORA - (Indirect),Y
12 - Future Expansion
13 - Future Expansion
14 - Future Expansion
15 - ORA - Zero Page,X
16 - ASL - Zero Page,X
17 - Future Expansion
18 - CLC
19 - ORA - Absolute,Y
1A - Future Expansion
1B - Future Expansion
1C - Future Expansion
1D - ORA - Absolute,X
1E - ASL - Absolute,X
1F - Future Expansion
2Ø - JSR
21 - AND - (Indirect,X)
22 - Future Expansion
23 - Future Expansion
24 - BIT - Zero Page
25 - AND - Zero Page
26 - ROL - Zero Page
27 - Future Expansion
28 - PLP
29 - AND - Immediate
2A - ROL - Accumulator
2B - Future Expansion
2C - BIT - Absolute
2D - AND - Absolute
2E - ROL - Absolute
2F - Future Expansion
3Ø - BMI
31 - AND - (Indirect),Y
32 - Future Expansion
33 - Future Expansion
34 - Future Expansion
35 - AND - Zero Page,X
36 - ROL - Zero Page,X
37 - Future Expansion
38 - SEC
39 - AND - Absolute,Y
3A - Future Expansion
3B - Future Expansion
3C - Future Expansion
3D - AND - Absolute,X
3E - ROL - Absolute,X
3F - Future Expansion
4Ø - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
5Ø - BVC
51 - EOR - (Indirect),Y
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion
6Ø - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
7Ø - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion
8Ø - Future Expansion
81 - STA - (Indirect,X)
82 - Future Expansion
83 - Future Expansion
84 - STY - Zero Page
85 - STA - Zero Page
86 - STX - Zero Page

87 - Future Expansion
88 - DEY
89 - Future Expansion
8A - TXA
8B - Future Expansion
8C - STY - Absolute
8D - STA - Absolute
8E - STX - Absolute
8F - Future Expansion
9Ø - BCC
91 - STA - (Indirect),Y
92 - Future Expansion
93 - Future Expansion
94 - STY - Zero Page,X
95 - STA - Zero Page,X
96 - STX - Zero Page,Y
97 - Future Expansion
98 - TYA
99 - STA - Absolute,Y
9A - TXS
9B - Future Expansion
9C - Future Expansion
9D - STA - Absolute,X
9E - Future Expansion
9F - Future Expansion
AØ - LDY - Immediate
A1 - LDA - (Indirect,X)
A2 - LDX - Immediate
A3 - Future Expansion
A4 - LDY - Zero Page
A5 - LDA - Zero Page
A6 - LDX - Zero Page
A7 - Future Expansion
A8 - TAY
A9 - LDA - Immediate
AA - TAX
AB - Future Expansion
AC - LDY - Absolute
AD - LDA - Absolute
AE - LDX - Absolute
AF - Future Expansion
BØ - BCS
B1 - LDA - (Indirect),Y
B2 - Future Expansion
B3 - Future Expansion
B4 - LDY - Zero Page,X
B5 - LDA - Zero Page,X
B6 - LDX - Zero Page,Y
B7 - Future Expansion
B8 - CLV
B9 - LDA - Absolute,Y
BA - TSX
BB - Future Expansion
BC - LDY - Absolute,X
BD - LDA - Absolute,X
BE - LDX - Absolute,Y
BF - Future Expansion
CØ - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
DØ - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP - Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion
EØ - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
FØ - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,X
FF - Future Expansion

GESTIONE DELLA MEMORIA SUL COMMODORE 64

Il COMMODORE 64 ha 64K byte di RAM, oltre a 20K di ROM contenenti il BASIC, il SIstema Operativo ed il set di caratteri standard. Una porzione di memoria di 4K e' inoltre destinata ai dispositivi di accesso di I/O. Come puo' essere possibile tutto cio' su un computer dotato di un bus indirizzi a 16 bit, capace solamente di indirizzare, in codizioni normali, 64K ?

Il segreto sta nel microprocessore 6510: su questo circuito si trova una porta di input/output che serve a controllare la RAM, la ROM o l'I/O che compaiono in determinate parti della memoria del sistema. Questa porta e' usata anche per controllare il DATASSETTE(TM), percio' e' importante considerare solamente i bit appropriati.

La porta di I/O del 6510 compare nella locazione 1. Il registro direzione dati per questa porta compare nella locazione 0. La porta e' controllata come ogni altra porta di input/output presente nel sistema...il registro direzione dati controlla se un certo bit e' di ingresso o di uscita, e se il trasferimento di dati avviene attraverso questa porta.

Nella porta di controllo del 6510, le righe sono definite nel modo seguente:

NOME	BIT	DIREZIONE	DESCRIZIONE
LORAM	0	OUTPUT	Controllo per RAM/ROM da $A000 a $BFFF (BASIC)
HIRAM	1	OUTPUT	Controllo per RAM/ROM da $E000 a $FFFF (KERNAL)
CHAREN	2	OUTPUT	Controllo per I/O/ROM da $D000 a $DFFF
	3	OUTPUT	Linea di scrittura Cassette
	4	INPUT	Senso interruttori Cassette
	5	OUTPUT	Controllo motore Cassette

Il valore appropriato del registro direzione dati e' il seguente:

BITS	5	4	3	2	1	0
	1	0	1	1	1	1

dove 1 indica output e 0 input.

La somma di tali cifre binarie e' 47 decimale. Il COMMODORE 64 imposta automaticamente il registro direzione dati a questo valore.

In generale, le linee di controllo eseguono la funzione riportata nella loro descrizione. Occasionalmente, si usa una combinazione di linee di controllo per ottenere una particolare configurazione della memoria.

LORAM (bit 0) puo' essere generalmente pensata come una linea di controllo che inserisce o esclude dall'area indirizzabile del microprocessore la ROM di 8K del BASIC. Per le operazioni BASIC questa linea e' normalmente ALTA. Se questa linea e' predisposta BASSA, la ROM del BASIC scompare dalla mappa di memoria ed e' rimpiazzata dagli 8K byte di RAM che vanno dalla locazione $A000 alla locazione $BFFF.

HIRAM (bit 1) puo' essere generalmente pensato come una linea di controllo che inserisce o esclude dall'area indirizzabile del microprocessore la ROM di 8K bytes del KERNAL. Per le operazioni in BASIC questa linea e' normalmente ALTA. Se questa linea viene predisposta BASSA, la ROM del KERNAL scompare dalla mappa di memoria ed e' rimpiazzata dagli 8K byte di RAM che vanno dalla locazione $E000 alla locazione $FFFF.

CHAREN (bit 2) e' usata solamente per inserire o escludere dall'area indirizzabile del microprocessore la ROM di 4K byte del generatore di caratteri. Dal punto di vista del processore, la ROM dei caratteri occupa la stessa area indirizzabile dei dispositivi di I/O ($D000-$DFFF). Quando la linea CHAREN e' impostata a 1 (condizione normale), i dispositivi di I/O appaiono nell'area indirizzabile del microprocessore e la ROM dei caratteri non e' piu' accessibile. Quando il bit CHAREN e' azzerato, la ROM dei caratteri compare nell'area indirizzabile del microprocessore e i dispositivi di I/O non sono accessibili (il microprocessore ha bisogno di accedere alla ROM dei caratteri solo quando scarica nella RAM l'insieme dei caratteri della ROM. Per questo e' consigliabile vedere nel Capitolo riservato alla GRAFICA la Sezione dei CARATTERI PROGRAMMABILI). In certe configurazioni di memoria, CHAREN puo' essere sovrapposto da un'altra linea di controllo. Senza i dispositivi di I/O, CHAREN non ha alcun effetto sulla configurazione della memoria. La RAM appare invece nelle locazioni da $D000 a $DFFF.

NOTA: In ciascuna mappa di memoria contenente la ROM, un'istruzione WRITE (o POKE) rivolta ad una locazione ROM provoca la memorizzazione dei dati RAM "sotto" la ROM. La scrittura in una locazione ROM comporta la memorizzazione dei dati nella RAM "riservata". Questo permette, ad esempio, di mantenere uno schermo ad alta risoluzione sotto il controllo di una ROM, e di poterlo modificare riportandolo nello spazio di indirizzamento del processore. Normalmente, una READ ad una locazione ROM riporta il contenuto della ROM, ma non della RAM "riservata".

Indirizzi	Contenuto	Dimensione
E000-FFFF	ROM del KERNAL oppure RAM	8K
D000-DFFF	I/O oppure RAM oppure ROM carattere	4K
C000-CFFF	RAM	4K
A000-BFFF	ROM BASIC oppure RAM oppure innesto ROM	8K
8000-9FFF	RAM oppure innesto ROM	8K
4000-7FFF	RAM	16K
0000-3FFF	RAM	16K

DESCRIZIONE DELLE LOCAZIONI DI I/O

D000-D3FF	VIC (Controllore Video)	1K Byte
D400-D7FF	SID (Sintetizzatore del Suono)	1K Byte
D800-DBFF	RAM colore	500 Byte
DC00-DCFF	CIA 1 (Tastiera)	256 Byte
DD00-DDFF	CIA 2 (Bus seriale, Porta Utente/RS-232)	256 Byte
DE00-DEFF	Presa aperta di I/O #1 (Abilitatore CP/M)	256 Byte
DF00-DFFF	Presa aperta di I/O #2 (Disk)	256 Byte

Le due prese aperte di I/O sono state previste per scopi generali dell'Utente e per cartucce speciali di I/O (come IEEE); sperimentalmente, sono state predisposte per supportare la cartuccia Z-80 (opzione CP/M), e per fare da interfaccia ad un sistema di dischi ad alta velocita' e di basso costo.
Il sistema provvede allo "start automatico" dei programmi contenuti nella Cartuccia Espansione del COMMODORE 64. Il programma della cartuccia ha inizio se i primi 9 byte della cartuccia ROM a partire dalla locazione 32768 ($8000 HEX) contengono dati specifici: i primi due byte devono contenere il vettore dello Start a Freddo, usato dal programma della cartuccia; i due byte successivi, che partono da 32770 ($8002 HEX), devono contenere il vettore dello Start a Caldo, usato dal programma della cartuccia; i tre byte successivi contengono le lettere CBM, con il bit 7 impostato in ciascuna lettera; infine, gli ultimi due byte contengono la cifra 80 codificata in PET ASCII.

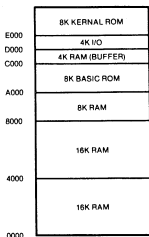
MAPPE DELLA MEMORIA DEL COMMODORE 64

Le seguenti tabelle riportano la lista delle varie configurazioni di memoria disponibili sul COMMODORE 64, gli stati delle linee di controllo che selezionano ciascuna mappa di memoria e l'applicazione particolare di ciascuna di esse.

1) Mappa di default della memoria BASIC;fornisce il BASIC 2.0 e 38K contigui di RAM Utente.

LORAM = 1 HIRAM = 1 GAME = 1 EXROM = 1

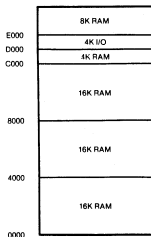
(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	8K BASIC ROM
A000	8K RAM
8000	16K RAM
4000	16K RAM
0000	

2) Mappa comprendente 60K di RAM e di dispositivi di I/O. Le procedure di I/O di accesso ai dischi sono a carico dell'Utente (devono essere scritte da quest'ultimo)

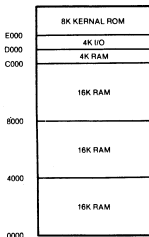
LORAM = 1 HIRAM = 0 GAME = 1 EXROM = X,
LORAM = 1 HIRAM = 0 GAME = 0 EXROM = 0

(La ROM carattere non e' accessibile dalla CPU in questa mappa)

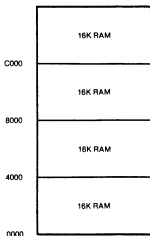
(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
	8K RAM
E000	4K I/O
D000	4K RAM
C000	16K RAM
8000	16K RAM
4000	16K RAM
0000	

3) Mappa costruita per l'uso dei programmi "softload" (compreso il CP/M); comprende 52K byte contigui di RAM Utente, i dispositivi di I/O e le routine di I/O per l'accesso ai dischi

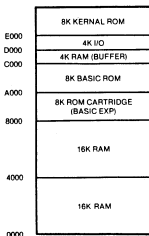
LORAM = 0 HIRAM = 1 GAME = 1 EXROM = X
(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM
C000	16K RAM
8000	16K RAM
4000	16K RAM
0000	

4) Mappa costruita per l'accesso a tutti i 64K byte RAM. I dispositivi di I/O devono essere riportati nell' area indirizzabile del processore per ogni operazione di I/O

LORAM = 0 HIRAM = 0 GAME = 1 EXROM = X
LORAM = 0 HIRAM = 0 GAME = X EXROM = 0
(X=non usato, 0=basso, 1=alto)

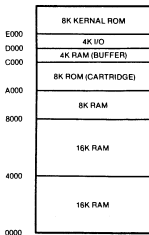
Indirizzo	Contenuto
C000	16K RAM
8000	16K RAM
4000	16K RAM
0000	16K RAM

5) Mappa rappresentante la configurazione standard di un sistema BASIC con una ROM espansa BASIC. Fornisce 32K RAM contigui Utente e fino a 8K bytes di "arricchimento" BASIC

LORAM = 1 HIRAM = 1 GAME = 0 EXROM = 0
(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
E000	8K KERNAL ROM
D000	4K I/O
C000	4K RAM (BUFFER)
A000	8K BASIC ROM
8000	8K ROM CARTRIDGE (BASIC EXP)
4000	16K RAM
0000	16K RAM

6) Mappa rappresentante una configurazione di 40K byte contigui di RAM Utente; mette a disposizione fino a 8K byte di innesto ROM per applicazioni speciali basate sulla ROM, che non richiedono il BASIC

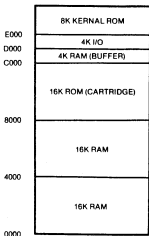
LORAM = 0 HIRAM = 1 GAME = 0 EXROM = 0

(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	8K ROM (CARTRIDGE)
A000	8K RAM
8000	16K RAM
4000	16K RAM
0000	

7) Questa mappa mette a dispsizione 32K byte contigui di RAM Utente, lasciando fino a 16K bytes di innesto ROM per applicazioni speciali basate sulla ROM non richiedenti il BASIC (word processor, altri linguaggi, ecc.)

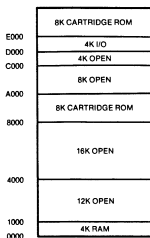
LORAM = 1 HIRAM = 1 GAME = 0 EXROM = 0

(X=non usato, 0=basso, 1=alto)

Indirizzo	Contenuto
	8K KERNAL ROM
E000	4K I/O
D000	4K RAM (BUFFER)
C000	16K ROM (CARTRIDGE)
8000	16K RAM
4000	16K RAM
0000	

8) Mappa ULTIMAX per videogiochi. Si noti che, se richiesto, si puo' accedere dall'esterno del COMMODORE 64 a 2K byte per ULTIMAX, senza considerare alcuna RAM della cartuccia

LORAM = X HIRAM = X GAME = 0 EXROM = 1

(X=non usato, 0=basso, 1=alto)

Indirizzo	Area
	8K CARTRIDGE ROM
E000	4K I/O
D000	4K OPEN
C000	8K OPEN
A000	8K CARTRIDGE ROM
8000	16K OPEN
4000	12K OPEN
1000	4K RAM
0000	

IL KERNAL

Uno dei problemi che deve essere affrontato dai programmatori nel campo dei microcomputer e' la questione di che cosa fare quando la casa costruttrice modifica il Sistema Operativo del computer. I programmi in linguaggio macchina, che richiedono molto tempo per essere sviluppati, potrebbero non funzionare piu', costringendo ad apportare modifiche complesse al programma. Per rendere meno oneroso questo problema, la Commodore ha sviluppato un metodo chiamato KERNAL, che interviene in difesa degli scrittori di software.

Praticamente, il KERNAL e' una TABELLA DEI SALTI standardizzata rivolta all'input, all'output ed alle routine di gestione della memoria del Sistema Operativo. Le locazioni di ciascuna routine che si trova in ROM puo' essere cambiata quando viene rinnovato il sistema, ma la tavola dei salti KERNAL sara' modificata continuamente per adattarsi. Modificare le sottoprocedure scritte in Linguaggio Macchina e' molto piu' veloce se esse usano solamente la routine di sistema in ROM, tramite il KERNAL.

Il KERNAL costituisce il Sistema Operativo del COMMODORE 64. Tutto l'input, l'output e la gestione della memoria e' controllato dal KERNAL. Per semplificare i programmi in Linguaggio Macchina, e per essere sicuri che la futura versione del Sistema Operativo del COMMODORE 64 non renda inutilizzabili i programi scritti in Linguaggio Macchina, il KERNAL comprende una tabella dei salti. Traendo vantaggio dalle 39 routine di input/output e da altre "utilities" disponibili sulla tabella, non solo si risparmia tempo, ma e' anche piu' facile trasferire i programmi dal COMMODORE 64 ad un altro computer.

La tabella dei salti ha la sua locazione nell'ultima pagina di memoria, nella Memoria a Sola Lettura (ROM).

Per usare la tabella dei salti del KERNAL e' necessario, per prima cosa, impostare i parametri necessari al funzionamento della procedura KERNAL. Quindi occorre saltare alla sottoprocedura, tramite l'istruzione JSR, nel punto esatto della tabella dei salti del KERNAL. Dopo che il KERNAL ha esaurito le sue funzioni, riporta il controllo al programma in Linguaggio Macchina. A seconda di quale routine del KERNAL si sta usando, certi registri possono ritornare parametri al programma. I registri propri di ciascuna routine del KERNAL possono essere individuati nelle singole descrizioni delle sottoprocedure del KERNAL.

A questo punto, una buona domanda potrebbe essere la seguente: "Perche' usare la tavola dei salti ? Perche' JSR non va ad interessare direttamente la sottoprocedura del KERNAL ?". Si usa la tabella dei salti perche',se il KERNAL o il BASIC vengono modificati, i programmi in linguaggio macchina possono essere ancora validi. In un futuro Sistema Operativo, le routine potranno trovarsi allocate in un'altra parte della mappa della memoria...ma la tavola dei salti funzionera' ancora bene !

ATTIVITÀ DI INIZIALIZZAZIONE DEL KERNAL

1) All'inizializzazione, il KERNAL riattiva innanzitutto il puntatore allo stack, poi azzera il modo decimale.
2) Il KERNAL passa poi a controllare la presenza di una cartuccia per l'avviamento automatico della ROM, nella locazione $8000 HEX (32768 decimale). Se questa e' presente, l'inizializzazione normale viene sospesa ed il controllo viene trasferito al codice della cartuccia; in caso contrario, continua il normale sistema di inizializzazione.
3) Successivamente il KERNAL inizializza tutti i dispositivi di input/output, ed anche il bus seriale. Entrambi i circuiti CIA 6526 sono impostati ai valori richiesti per la scansione della tastiera; viene attivato il timer a 60 Hz. Il circuito SID viene azzerato. Viene selezionata la mappa della memoria BASIC e viene spento il motore del registratore.
4) Il KERNAL esegue a questo punto un test sulla RAM, impostando i puntatori in cima ed in fondo alla memoria. Viene inizializzata anche la Pagina Zero, e viene impostato il buffer del nastro.
   Il test sulla RAM e' una procedura non distruttiva, che parte dalla locazione $0300 e si muove verso l'alto. Una volta raggiunta la prima locazione non RAM, la cima della RAM si ritrova il puntatore impostato. La base della memoria e' sempre impostata alla locazione $0800, mentre la memoria dello schermo parte dalla locazione $0400.
5) Infine, il KERNAL esegue le altre attivita'. I vettori di I/O vengono impostati a valori di default. La tabella dei salti indiretti viene fissata nella memoria di base. Lo schermo viene azzerato, e vengono riattivate tutte le variabili dell'editor di schermo. Quindi viene usato l'indirizzo posto nella locazione $A000 per dare il via al BASIC.

COME USARE IL KERNAL

Quando si scrivono programmi in Linguaggio Macchina e' conveniente usare le routine riguardanti l'input/output che fanno gia' parte del Sistema Operativo, l'accesso al clock di sistema, la gestione della memoria ed altre operazioni simili. Dato il facile accesso al Sistema Operativo, che rende piu' spedita la programmazione in Linguaggio Macchina, la riscrittura di tali routine non risulta altro che uno sforzo inutile.

Come si e' gia' detto, il KERNAL e' una tabella dei salti, costituita da un insieme di istruzioni JMP che consentono di saltare alle varie routine del Sistema Operativo.

Per usare una procedura del KERNAL e' necessario eseguire per prima cosa tutte le operazioni richieste dalla procedura stessa. Ad esempio, se una procedura del KERNAL richiede di essere chiamata prima di un'altra, essa deve essere chiamata. Se la procedura si aspetta l'inserimento di un numero nell'accumulatore, allora questo numero si deve trovare li'; diversamente, la procedura ha poche probabilita' di funzionare nel modo previsto.

Dopo essersi preparati, occorre chiamare la procedura per mezzo dell'istruzione JSR. Tutte le procedure del KERNAL a cui si puo'

accedere sono strutturate come SOTTOPROCEDURE, e devono terminare con un'istruzione RTS. Quando la procedura del KERNAL ha terminato il suo compito, il controllo ritorna all'istruzione successiva alla JSR.

Molte procedure del KERNAL restituiscono i codici d'errore nella "Status Word", oppure nell'accumulatore, se insorgono problemi durante il loro svolgimento. La buona pratica della programmazione ed il successo dei programmi scritti in Linguaggio Macchina richiedono un trattamento adeguato di queste procedure: ignorare un ritorno d'errore potrebbe causare il fallimento del resto del programma.

Quando si vuole usare il KERNAL si devono eseguire i tre semplici passi seguenti:

1) Avviamento
2) Chiamare la routine
3) Gestire l'errore

Nella descrizione delle routine del KERNAL si usano le seguenti convenzioni:

NOME DELLA FUNZIONE - Nome della routine del KERNAL.

INDIRIZZO DI CHIAMATA - Indirizzo di chiamata della routine del KERNAL, espresso in esadecimale.

REGISTRI DI COMUNICAZIONE - I registri di questa voce vengono usati per passare/ritornare i parametri alla/dalla rotine del KERNAL.

ROUTINE DI PREPARAZIONE - Alcune routine del KERNAL richiedono che siano passati i dati prima che esse possano operare. Le routine necessarie sono elencate qui di seguito.

RITORNO DELL'ERRORE - Un ritorno da una routine del KERNAL con il riporto impostato indica che durante l'elaborazione si e' verificato un errore. Il numero di tale errore e' contenuto nell'accumulatore.

RICHIESTE DELLO STACK - Numero attuale di byte dello stack usati dalla routine del KERNAL.

REGISTRI INTERESSATI - Tutti i registri usati dalla routine del KERNAL sono riportati qui di seguito.

DESCRIZIONE - Un breve elenco delle funzioni svolte dalle routine del KERNAL e' riportato qui di seguito.

La seguente e' la lista delle routine del KERNAL.

ROUTINE DEL KERNAL RICHIAMABILI DALL'UTENTE

NOME	INDIRIZZO		FUNZIONE
	HEX	DECIMALE	
ACPTR	$FFA5	65445	Accetta un byte dalla porta seriale
CHKIN	$FFC6	65478	Apre il canale di input
CHKOUT	$FFC9	65481	Apre il canale di output
CHRIN	$FFCF	65487	Accetta un carattere dal canale
CHROUT	$FFD2	65490	Immette un carattere nel canale
CIOUT	$FFA8	65448	Trasferisce un byte alla porta seriale
CINT	$FF81	65409	Inizializza l'editor di schermo
CLALL	$FFE7	65511	Chiude tutti i canali ed i files
CLOSE	$FFC3	65475	Chiude un file logico specifico
CLRCHN	$FFCC	65484	Chiude i canali di input e di output
GETIN	$FFE4	65508	Prende il carattere dalla coda (buffer) della tastiera
IOBASE	$FFF3	65523	Ritorna l'indirizzo di base dei dispositivi di I/O
IOINIT	$FF84	65412	Inizializza l'I/O
LISTEN	$FFB1	65457	Dispone a RICEVENTE i dispositivi sul bus seriale
LOAD	$FFD5	65493	Carica la RAM da un dispositivo
MEMBOT	$FF9C	65436	Legge/imposta la base della memoria
MEMTOP	$FF99	65433	Legge/imposta la cima della memoria
OPEN	$FFC0	65472	Apre un file logico
PLOT	$FFF0	65520	Legge/imposta la posizione X, Y del cursore
RAMTAS	$FF87	65415	Inizializza la RAM, alloca il buffer del nastro, imposta lo schermo a $0400
RDTIM	$FFDE	65502	Legge il clock del tempo
READST	$FFB7	65463	Legge la parola di stato di I/O
RESTOR	$FF8A	65418	Ripristina il vettore di default di I/O
SAVE	$FFD8	65496	Salva la RAM su un dispositivo
SCNKEY	$FF9F	65439	Fa la scansione della tastiera
SCREEN	$FFED	65517	Ritorna il sistema di coordinate X, Y di schermo
SECOND	$FF93	65427	Invia l'indirizzo secondario dopo RICEZIONE
SETLFS	$FFBA	65466	Imposta gli indirizzi primario, secondario e logico
SETMSG	$FF90	65424	Controlla i messaggi del KERNAL
SETNAM	$FFBD	65469	Imposta il nome del file
SETTIM	$FFDB	65499	Imposta il clock del tempo
SETTMO	$FFA2	65442	Imposta il supero tempo sul bus seriale
STOP	$FFE1	65505	Termina la scansione della tastiera
TALK	$FFB4	65460	Imposta a TRASMITTENTE il dispositivo del bus seriale
TKSA	$FF96	65430	Invia l'indirizzo secondario dopo TRASMISSIONE
UDTIM	$FFEA	65514	Incrementa il clock del tempo
UNLSN	$FFAE	65454	Imposta il bus seriale a NON-RICEVENTE
UNTLK	$FFAB	65451	Imposta il bus seriale a NON-TRASMITTENTE
VECTOR	$FF8D	65421	Legge/imposta il vettore di I/O

B.1 - Nome della funzione: ACPTR

Scopo: Prende i dati dal bus seriale
Indirizzo di chiamata: $FFA5 (HEX), 65445 (decimale)
Registri di comunicazione: .A
Procedura di preparazione: TALK, TKSA
Errori di ritorno: Vedere READST
Richiesta dello stack: 13
Registri interessati: .A, .X

Descrizione: - Questa routine deve essere usata quando si vogliono ricevere informazioni da un dispositivo sul bus seriale, come ad esempio un disco. Questa routine preleva dal bus seriale un byte del dato usando l'"Handshacking". Il dato viene riportato nell'accumulatore. Per preparare questa routine e' necessario chiamare la routine TALK, che ordina al dispositivo sul bus seriale di trasmettere dati attraverso il bus. Se il dispositivo ha bisogno di un comando secondario, quest'ultimo deve essere inviato per mezzo della routine del KERNAL TKSA, prima che la routine in esame venga chiamata. Gli errori sono riportati nella parola di stato, che viene letta dalla routine READST.

Come si usa:

0) Ordinare a un dispositivo sul bus seriale di predisporsi all'invio dati al COMMODORE 64 (usare le routine del KERNAL TALK e TKSA)
1) Chiamare queste routine (usando JSR)
2) Memorizzare, oppure usare, i dati

ESEMPIO:

```
;PRENDE UN BYTE DAL BUS
  JSR ACPTR
  STA DATA
```

B.2 – Nome della funzione: CHKIN

Scopo: Apre un canale di input
Indirizzo di chiamata: $FFC6 (HEX), 65478 (decimale)
Registri di comunicazione: .X
Procedure di preparazione: (OPEN)
Errori di ritorno:
Richiesta dello stack: Nessuna
Registri interessati: .A, .X

Descrizione: - Ogni file logico che e' gia' stato aperto dalla routine del KERNAL OPEN puo' essere definito, da questa routine, come canale di input Normalmente, il dispositivo sul canale deve essere un dispositivo di input; diversamente, si verifica un errore e la routine "abortisce"
All'atto della ricezione di dati provenienti da qualunque altra parte diversa dalla tastiera, prima di usare le routine del KERNAL CHRIN o GETIN si deve chiamare questa routine. Se si vuole usare l'input da tastiera, e non sono aperti altri canali di input, le chiamate a questa routine ed alla routine OPEN non sono necessarie.

Quando si usa questa routine in congiunzione ad un dispositivo sul bus seriale, essa provvede ad inviare automaticamente sul bus l'indirizzo di chiamata (e l'indirizzo secondario, se specificato nella routine OPEN).

Come si usa:

0) Aprire (OPEN) il file logico (se necessario: vedere descrizione precedente).
1) Caricare il registro .X con il numero di file logico che deve essere usato.
2) Chiamare questa routine (usando il comando JSR)

Possibili errori:

#3 : File non aperto
#5 : Dispositivo non presente
#6 : Il file non e' un file di input

ESEMPIO:

```
;SI PREPARA PER UN INPUT PROVENIENTE DAL FILE LOGICO 2
LDX #2
JSR CHKIN
```

B.3 – Nome della funzione: CHKOUT

Scopo: Apre un canale di output
Indirizzo di chiamata: $FFC9 (HEX), 65481 (decimale)
Registri di comunicazione: .X
Procedure di preparazione: (OPEN)
Errori di ritorno: 0, 3, 5, 7 (vedere READST)
Richiesta dello stack: 4+
registri interessati: .A, .X

Descrizione: – Ciascun numero di file logico, creato dalla routine del KERNAL OPEN, puo' essere definito come un canale di output. Naturalmente, il dispositivo che si intende usare per aprire un canale deve essere di output, altrimenti si verifica un errore e la routine "abortisce".
Questa routine deve essere chiamata prima che qualsiasi dato venga inviato su un dispositivo di output, a meno che quest'ultimo non sia lo schermo del COMMODORE 64. In questo caso, se non ci sono altri canali di output gia' predisposti, la chiamata a questa routine ed alla routine OPEN non e' necessaria.
Quando si usa questa routine per aprire un canale di un dispositivo sul bus seriale, essa invia automaticamente l'indirizzo di RICEZIONE specificato dalla routine OPEN (e l'indirizzo secondario, se specificato).

Come si usa:

RICORDARE: Questa routine NON E' NECESSARIA per inviare i dati sullo schermo.

0) Usare la routine OPEN del KERNAL per specificare il numero di file logico, l'indirizzo di RICEZIONE e, se specificato, l'indirizzo secondario.
1) Caricare il registro .X con il numero del file logico usato nella istruzione di apertura.
2) Chiamare questa routine usando JSR

ESEMPIO:

```
;DEFINISCE IL FILE LOGICO 3 COME CANALE DI OUTPUT
LDX #3
JSR CHKOUT
```

Possibili errori:

#3 : File non aperto
#5 : Dispositivo non presente
#7 : File non di output

B.4 – Nome della funzione: CHRIN

Scopo: Preleva un carattere dal canale di input
Indirizzo di chiamata: $FFCF (HEX), 65487 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: (OPEN, CHKIN)
Errori di ritorno: 0 (vedere READST)
Richiesta dello stack: 7+
Registri interessati: .A, .X

Descrizione: – Questa routine preleva un byte del dato da un canale gia' predisposto come canale di input dalla routine CHKIN del KERNAL. Se CHKIN non e' stata usata per definire un altro canale di input, allora si prevede che i dati provengano dalla tastiera. Il byte del dato viene sistemato nell'accumulatore. Dopo la chiamata, il canale rimane aperto.
L'input da tastiera e' gestito in modo particolare. Per prima cosa, viene attivato il cursore, che continua a lampeggiare finche' non viene digitato sulla tastiera un ritorno carrello. Tutti i caratteri che si trovano sulla linea (fino a 88) vengono memorizzati nel buffer di input del BASIC, per poi essere ripresi uno alla volta da questa routine. Quando viene chiamato il ritorno carrello, tutta la linea e' stata elaborata. Alla chiamata successiva di questa routine, questo processo, a partire dal lampeggiamento del cursore, viene eseguito di nuovo.

Come si usa:

DA TASTIERA
1) Recuperare un byte del dato tramite la chiamata a questa routine
2) Memorizzare il byte del dato
3) Verificare se si tratta dell'ultimo byte del dato
4) Se non lo e', ritornare al passo 1

ESEMPIO:

```
    LDY $#00      ;PREPARA IL REGISTRO .Y ALLA MEMORIZZAZIONE DEL DATO
RD  JSR CHRIN
    STA DATA,Y    ;MEMORIZZA L'Y-ESIMO BYTE NELLA Y-ESIMA LOCAZIONE
                   DELL'AREA DATI
    INY
    CMP #CR       ;E' UN RITORNO CARRELLO?
    BNE RD        ;SE NON LO E', PRENDI UN ALTRO BYTE DEL DATO
```

DA ALTRI DISPOSITIVI:

0) Usare le routine OPEN e CHKIN del KERNAL
1) Chiamare questa routine (usando l'istruzione JSR)
2) Memorizzare il dato

ESEMPIO:

```
JSR CHRIN
STA DATA
```

B.5 – Nome della funzione: CHROUT

Scopo: Invia in output un carattere
Indirizzo di chiamata: $FFD2 (HEX), 65490 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: (CHKOUT, OPEN)
Errori di ritorno: 0 (vedere READST)
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: – Questa routine invia in output un carattere su un canale gia' aperto. Prima di chiamare questa routine, usare le routine OPEN e CHKOUT per predisporre il canale di output. Se la chiamata non viene effettuata, il dato viene inviato ad un dispositivo di output standard (il numero 3, cioe' lo schermo). Prima di chiamare questa routine, il byte del dato viene caricato nell'accumulatore, per poi essere inviato al dispositivo di output specificato. Dopo la chiamata, il canale rimane aperto.

NOTA: Questa routine deve essere trattata con cura quando si intenda inviare il dato ad uno specifico dispositivo seriale, poiche' il dato viene inviato a tutti i canali di output aperti sul bus. Tutti i canali di output aperti sul bus seriale, oltre a quello inteso per la destinazione, devono essere chiusi tramite una chiamata alla routine CLRCHN del KERNAL, a meno che non si desideri diversamente.

Come si usa:

0) Usare la routine CHKOUT del KERNAL, se necessario (vedere la descrizione sopra).
1) Caricare nell'accumulatore il dato da inviare in output.
2) Chiamare questa routine.

ESEMPIO:

```
;DUPLICATO DELL'ISTRUZIONE BASIC CMD 4,"A"
  LDX #4          ;FILE LOGICO #4
  JSR CHKOUT      ;APRE IL CANALE DI OUTPUT
  LDA #'A
  JSR CHROUT      ;INVIA IL CARATTERE
```

B.6 – Nome della funzione: CIOUT

Scopo: Trasmette un byte sul bus seriale
Indirizzo di chiamata: $FFA8 (HEX), 65448 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: LISTEN,[SECOND]
Errori di ritorno: vedere READST
Richiesta dello stack: 5
Registri interessati: nessuno

Descrizione: – Questa routine e' usata per inviare informazioni ai dispositivi sul bus seriale. Una chiamata a questa routine inserisce il byte del dato sul bus seriale usando l'"Handshacking" seriale. Prima di chiamare questa routine, e' necessario chiamare la routine LISTEN del KERNAL, che ordina ad un dispositivo sul bus seriale di tenersi pronto alla ricezione di dati (se un dispositivo ha bisogno di un indirizzo secondario, si deve inviare anche questo, tramite la routine SECOND del KERNAL). L'accumulatore viene caricato con un byte che viene trasmesso come dato sul bus seriale. Un dispositivo deve essere predisposto a ricevente, altrimenti la parola di stato rileva un "fuori sincronismo". Questa routine "bufferizza" sempre un carattere (la routine, cioe', trattiene il dato precedente a quello ritornato). Percio', quando si chiama la routine UNLSN del KERNAL per porre fine alla trasmissione dei dati, il carattere "bufferizzato" viene inviato con una "End Or Identify" (EOI) impostata; successivamente, al dispositivo viene inviato il comando UNLSN.

Come si usa:

0) Usare la procedura LISTEN del KERNAL (se necessario, anche la procedura SECOND)
1) Caricare l'accumulatore con un byte del dato
2) Chiamare questa procedura per inviare il byte del dato

ESEMPIO:

```
  LDA #'X         ;INVIA UNA X AL BUS SERIALE
  JSR CIOUT
```

B.7 – Nome della funzione: CINT

Scopo: Inizializza l'editor di schermo ed il circuito di controllo del video 6567
Indirizzo di chiamata: $FF81 (HEX), 65409 (decimale)
Registri di comunicazione: nessuno
Procedure di preparazione: nessuna
Errori di ritorno: nessuno
Richiesta dello stack: 4
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine predispone il circuito di controllo del video 6567del COMMODORE 64 ad un'operazione normale. Viene inizializzato anche l'editor di schermo. Questa routine deve essere chiamata da una cartuccia programma del COMMODORE 64.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR CINT
JMP RUN        ;INIZIO DELL'ESECUZIONE
```

B.8 – Nome della funzione: CLALL

Scopo: Chiude tutti i files
Indirizzo di chiamata: $FFE7 (HEX), 65511 (decimale)
Registri di comunicazione: nessuno
Errori di ritorno: nessuno
Richiesta dello stack: 11
Registri interessati: .A, .X

Descrizione: – Questa routine chiude tutti i files aperti. Quando questa routine viene chiamata, i puntatori alla tabella dei file aperti vengono impostati daccapo a zero, chiudendo tutti i files. Inoltre, viene chiamata automaticamente la routine CLRCHN, che imposta daccapo i canali di I/O.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
    JSR CLALL    ;CHIUDE TUTTI I FILES E SELEZIONA I CANALI DI I/O DI
DEFAULT
    JMP RUN      ;INIZIO DELL'ESECUZIONE
```

B.9 – Nome della funzione: CLOSE

Scopo: Chiude un file logico
Indirizzo di chiamata: $FFC3 (HEX), 65475 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: 0, 240 (vedere READST)
Richiesta dello stack: 2+
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine serve per chiudere un file logico dopo che su questo file sono state completate tutte le operazioni di I/O. Questa routine viene chiamata dopo che si e' caricato l'accumulatore con il numero del file logico che deve essere chiuso (il numero e' lo stesso usato per la sua apertura, avvenuta tramite la routine OPEN).

Come si usa:

1) Caricare l'accumulatore con il numero del file logico che deve essere chiuso.
2) Chiamare questa routine.

ESEMPIO:

```
;CLOSE 15
LDA #15
JSR CLOSE
```

B.10 – Nome della funzione: CLRCHN

Scopo: Azzera i canali di I/O
Indirizzo di chiamata: $FFCC (HEX), 65484 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno:
Richiesta di stack: 9
Registri interessati: .A, .X

Descrizione: – Questa routine viene chiamata per azzerare tutti i canali aperti e per restituire ai canali di I/O il loro valore standard originale. Di solito, viene chiamata dopo aver aperto altri canali di I/O (come nastro o disco) e dopo averli usati per operazioni di I/O. Il dispositivo standard di input e' il numero 0 (tastiera), quello standard di output e' il numero 3 (schermo).
Se uno dei dispositivi da chiudere e' la porta seriale, prima di tutto viene inviato@un segnale di UNTLK per azzerare il canale di input, oppure un segnale di@UNLISTEN per azzerare il canale di output. Se questa routine non viene chiamata (ed il ricevitore (o i ricevitori) viene lasciato attivo sul bus seriale), lo stesso dato puo' essere ricevuto contemporaneamente da piu' di un dispositivo del COMMODORE 64. Un modo per trarre vantaggio da tutto cio' e' quello di ordinare alla stampante di COMUNICARE (TALK) e al disco di RICEVERE (LISTEN), permettendo cosi' la stampa diretta di un file su disco.
Questa routine viene chiamata direttamente durante l'esecuzione della routine CLALL del KERNAL.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.

ESEMPIO:

```
JSR CLRCHN
```

B.11 – Nome della funzione: GETIN

Scopo: Prende un carattere
Indirizzo di chiamata: $FFE4 (HEX), 655508 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: CHKIN, OPEN
Errori di ritorno: vedere READST
Richiesta dello stack: 7+
Registri interessati: .A (.X, .Y)

Descrizione: – Se il canale e' la tastiera, questa sottoprocedura rimuove un carattere dalla coda della tastiera e lo restituisce all'accumulatore sottoforma di valore in ASCII. Se la coda e' vuota, il valore che viene riportato nell'accumulatore e' zero. I caratteri sono inseriti automaticamente nella coda da una routine di scansione della tastiera, pilotata da interruzioni, che si chiama SCNKEY. Il buffer della tastiera puo' contenere un massimo di 10 caratteri. Quando il buffer e' pieno, i caratteri in eccedenza sono ignorati fino alla prossima rimozione di un carattere dalla coda. Se il canale e' l'RS-232, allora viene usato solamente il registro .A e viene ritornato un singolo carattere. Per il controllo della validita' si veda READST. Se il canale e' di tipo seriale, oppure e' il registratore o lo schermo, allora chiamare la routine BASIN.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.
2) Controllare se nell'accumulatore c'e' zero (buffer vuoto).
3) Elaborare i dati.

ESEMPIO:

```
;ATTENDE UN CARATTERE
WAIT JSR GETIN
CMP #0
BEQ WAIT
```

B.12 – Nome della funzione: IOBASE

Scopo: Definisce la pagina di memoria perl'I/O
Indirizzo di chiamata: $FFF3 (HEX), 65523 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno:
Richiesta dello stack: 2
Registri interessati: .X, .Y

Descrizione: – Questa routine imposta i registri X e Y all'indirizzo del segmento di memoria dove sono allocati i dispositivi di I/O mappati in memoria. Questo indirizzo puo' quindi essere usato come "offset" per accedere ai dispositivi di I/O mappati nella memoria del COMMODORE 64. L'"offset" rappresenta il numero delle locazioni dall'inizio della pagina sulle quali si vuole allocare il registro di I/O. Il registro .X contiene il byte basso dell'indirizzo, mentre il registro .Y contiene il byte alto dell'indirizzo.
Questa routine e' stata creata per garantire la compatibilta' tra il COMMODORE 64, il VIC 20 ed i prossimi modelli del COMMODORE 64. Se le locazioni per l'I/O, limitatamente ad un programma in Linguaggio Macchina, vengono assegnate tramite una chiamata a questa routine, allora risultano ancora compatibili con le prossime versioni del COMMODORE 64, del KERNAL e del BASIC.

Come si usa:

1) Chiamare questa routine usando l'istruzione JSR.
2) Memorizzare i registri .X e .Yin due locazioni consecutive
3) Caricare il registro .Y con l'"offset"
4) Accedere a quella locazione di I/O.

ESEMPIO:

```
;IMPOSTA A ZERO (INPUT) IL REGISTRO DIREZIONE DATI DELLA PORTA UTENTE
    JSR IOBASE
    STX POINT       ;IMPOSTA I REGISTRI BASE
    STY POINT+1
    LDY #2
    LDA #0          ;OFFSET PER DDR DELLA PORTA UTENTE
    STA (POINT), Y  ;IMPOSTA DDR A ZERO
```

B.13 – Nome della funzione: IOINIT

Scopo: Inizializza i dispositivi di I/O
Indirizzo di chiamata: $FF84 (HEX), 65412 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nesuna
Errori di ritorno:
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione. – Questa routine inizializza tutti i dispositivi e le routine di I/O e le rotine. Viene normalmente chiamata come parte della procedura di inizializzazione di una cartuccia programma del

COMMODORE 64.

ESEMPIO:

```
JSR IOINIT
```

B.14 – Nome della funzione: LISTEN

Scopo: Predispone il dispositivo sul bus seriale a ricezione
Indirizzo di chiamata: $FFB1 (HEX), 65457 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: Nessuno
Registri interessati: .A

Descrizione: – Questa routine ordina al dispositivo sul bus seriale di ricevere dati. Prima di effettuare la chiamata a questa routine, l'accumulatore deve essere caricato con un numero di dispositivo compreso fra 0 e 31. La funzione LISTEN esegue una OR su tutti i bit del numero per convertirlo ad un indirizzo di ascolto, e per trasmettere poi questo dato sul bus seriale sotto forma di comando. Il dispositivo specificato viene posto nella condizione di ricevere, ed e' quindi pronto ad accettare l'informazione.

Come si usa:

1) Caricare l'accumulatore con il numero del dispositivo a cui si desidera comandare di RICEVERE.
2) Chiamare questa routine usando l'istruzione JSR.

ESEMPIO:

```
;COMANDA AL DISPOSITIVO #8 DI RICEVERE
  LDA #8
  JSR LISTEN
```

B.15 – Nome della funzione: LOAD

Scopo: Carica da dispositivo in RAM
Indirizzo di chiamata: $FFD5 (HEX), 65493 (decimale)
Registri di comunicazione: .A, .X, .Y
Errori di ritorno: 0,4,5,8,9, READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine carica i byte del dato da qualsiasi dispositivo di input nella memoria del COMMODORE 64. Puo' essere usata anche per operazioni di verifica, confrontando il dato sul dispositivo con quello gia' in memoria e lasciando invariato il dato memorizzato in RAM. L'accumulatore viene impostato a 0 per un caricamento e a 1 per una verifica. Se il dispositivo di input e' aperto con indirizzo secondario 0, l'informazione contenuta nell'etichetta di testa

proveniente dal dispositivo viene ignorata. In questo caso, i registri .X e .Y devono contenere l'indirizzo di partenza per il caricamento. Se il dispositivo e' indirizzato con indirizzo secondario 1, 0 o 2, allora il dato viene caricato in memoria a partire dalla locazione di memoria specificata dalla testata. Questa routine ritorna l'indirizzo della piu' alta locazione RAM caricata.
Prima di effettuare la chiamata per questa routine, occorre chiamare le routine SETLFS e SETNAM del KERNAL.

NOTA: NON E' POSSIBILE caricare da tastiera (0), da RS-232 (2) o da schermo (3).

Come si usa:

0) Chiamare le routine SETLFS e SETNAM. Se si desidera un caricamento rilocato, usare la routine SETLFS per inviare un indirizzo secondario 0.
1) Impostare il registro .A a 0 per caricamento e a 1 per verifica.
2) Se si desidera un caricamento rilocato, occorre impostare i registri .X e .Y all'indirizzo di partenza per il caricamento.
3) Chiamare la routine usando l'istruzione JSR.

ESEMPIO:

```
          ;CARICA UN FILE DA REGISTRATORE
          LDA #DEVICE1       ;IMPOSTA IL NUMERO DI DISPOSITIVO
          LDX #FILENO        ;IMPOSTA IL NUMERO LOGICO DEL FILE
          LDY CMD1           ;IMPOSTA L'INDIRIZZO SECONDARIO
          JSR SETLFS
          LDA #NAME1-NAME    ;CARICA .A CON IL NUMERO DI  CARATTERI DEL
                             NOME DEL FILE
          LDX #<NAME         ;CARICA .X CON L'INDIRIZZO  DEL  NOME  DEL
FILE
          LDY #>NAME         ;CARICA .Y CON L'INDIRIZZO  DEL  NOME  DEL
FILE
          JSR SETNAM
          LDA #0             ;IMPOSTA L'INDICATORE A CARICAMENTO
          LDX #$FF           ;ALTERNA LA PARTENZA
          LDY #$FF
          JSR LOAD
          STX VARTAB         ;TERMINE CARICAMENTO
          JMP START
 NAME     .BYT 'FILE NAME'
 NAME1    ;
```

B.16 – Nome della funzione: MEMBOT

Scopo: Imposta la base della memoria
Indirizzo di chiamata: $FF9C (HEX), 65436 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: Nessuna
Registri interessati: .X, .Y

Descrizione: - Questa routine e' usata per impostare la base della memoria. Se il bit di riporto dell'accumulatore risulta impostato, allora, quando questa routine viene chiamata, nei registri .X e .Y viene riportato il valore del puntatore al byte piu' basso della RAM. Sul COMMODORE, 64 privo di espansioni di memoria, il valore iniziale di questo puntatore e' $0800 (2048 decimale). Se il bit di riporto dell'accumulatore e' a zero al momento della chiamata della routine, i valori dei registri .X e .Y vengono trasferiti rispettivamente al byte basso ed al byte alto del puntatore all'inizio della RAM.

Come si usa:

PER LEGGERE IL VALORE DALLA BASE DELLA RAM

1) Impostare il riporto
2) Chiamare questa routine

PER IMPOSTARE IL VALORE DELLA BASE DELLA RAM

1) Azzerare il riporto
2) Chiamare questa routine

ESEMPIO:

```
  ALZA LA BASE DELLA MEMORIA DI UNA PAGINA
     SEC        ;LEGGE LA BASE DELLA MEMORIA
     JSR MEMBOT
     INY
     CLC        ;IMPOSTA LA BASE DELLA MEMORIA AL NUOVO VALORE
     JSR MEMBOT
```

B.17 - Nome della funzione: MEMTOP

Scopo: Imposta la cima della RAM
Indirizzo di chiamata: $FF99 (HEX), 65433 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .X, .Y

Descrizione: - Questa routine e' usata per impostare la cima della RAM. Quando si chiama questa routine con il bit di riporto impostato, nei registri .X e .Y viene caricato il valore del puntatore alla cima della RAM. Se invece il bit di riporto dell'accumulatore e' azzerato, nella cima della RAM viene caricato il contenuto dei registri .X e .Y, cambiando cosi' la cima della memoria.

ESEMPIO:

```
 ;DISALLOCA IL BUFFER DELL'RS-232
   SEC          ;LEGGE LA CIMA DELLA MEMORIA
   JSR MEMTOP
   DEX
   CLC
   JSR MEMTOP   ;IMPOSTA LA NUOVA CIMA DELLA MEMORIA
```

B.18 – Nome della funzione: OPEN

Scopo: Apre un file logico
Indirizzo di chiamata: $FFC0 (HEX), 65472 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: SETLFS, SETNAM
Errori di ritorno: 1,2,4,5,6,240,READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine e' usata per aprire un file logico. Una volta predisposto, il file logico puo' essere usato per le operazioni di input/output. La maggior parte delle routine di I/O del KERNAL chiamano questa routine per creare il file logico su cui lavorare. Prima di usare questa routine, occorre chiamare le routine SETLFS e SETNAM del KERNAL.

Come si usa:

0) Usare la routine SETLFS.
1) Usare la routine SETNAM.
2) Chiamare questa routine.

ESEMPIO:

Implementazione dell'istruzione BASIC OPEN 15,8,15,"I/O"

```
        LDA #NAME2-NAME    ;LUNGHEZZA DEL NOME DEL FILE PER SETLFS
        LDY #>NAME         ;INDIRIZZO DEL NOME DEL FILE
        LDX #<NAME
        JSR SETNAM
        LDA #15
        LDX #8
        LDY #15
        JSR SETLFS
        JSR OPEN
NAME    .BYT 'I/O'
NAME2
```

B.19 - Nome della funzione: PLOT

Scopo: Imposta la locazione del cursore
Indirizzo di chiamata: $FFF0 (HEX), 65520 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: - Una chiamata a questa routine, con impostato l'indicatore di riporto dell'accumulatore, causa il caricamento della posizione attuale del cursore sullo schermo (espressa nelle coordinate X e Y) nei registri .X e .Y. Il numero di colonna (0...79) della posizione del cursore e' rappresentato da Y, mentre il numero di riga (0...24) della posizione del cursore e' rappresentato da X. Una chiamata con il bit di riporto azzerato posiziona il cursore nel punto di coordinate X,Y, come determinato dai registri .X e .Y

Come si usa:

PER LEGGERE LA LOCAZIONE DEL CURSORE

1) Impostare l'indicatore di riporto.
2) Chiamare la routine.
3) Prelevare la posizione di X e Y rispettivamente dai registri .X e Y

PER IMPOSTARE LA LOCAZIONE DEL CURSORE

1) Azzerare l'indicatore di riporto
2) Impostare i registri .X e .Y alla locazione di cursore desiderata.

ESEMPIO:

```
;POSIZIONA IL CURSORE NEL PUNTO DI COORDINATE (5,10)
;(RIGA=5, COLONNA=10)
  LDX #10
  LDY #5
  CLC
  JSR PLOT
```

B.20 - Nome della funzione: RAMTAS

Scopo: Esegue un test sulla RAM
Indirizzo di chiamata: $FF87 (HEX), 65415 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nesuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine e' usata per eseguire un test sulla RAM e per impostare di conseguenza i puntatori alla cima ed alla base della memoria; inoltre, azzera le locazioni da $0000 a $0101 e da

$0200 a $0BFF, alloca il buffer del registratore ed imposta la base dello schermo a $03FF. Normalmente, questa routine viene chiamata durante il processo di inizializzazione della cartuccia programma del COMMODORE 64.

ESEMPIO:

```
JSR RAMTAS
```

B.21 – Nome della funzione: RDTIM

Scopo: Legge il clock di sistema
Indirizzo di chiamata: $FFDE (HEX), 65502 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine e' usata per leggere il clock di sistema. La risposta del clock e' di 1/60 di secondo. La routine restituisce tre byte. L'accumulatore contiene il byte piu' significativo; il registro indice X contiene il byte piu' significativo seguente; il registro Y contiene l'ultimo byte piu' significativo.

ESEMPIO:

```
JSR RDTIM
STY TIME
STX TIME+1
STA TIME+2
. . .
TIME *=*+3
```

B.22 – Nome della funzione: READST

Scopo: Legge la parola di stato
Indirizzo di chiamata: $FFB7 (HEX), 65463 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: .A

Descrizione: Questa routine riporta nell'accumulatore lo stato attuale dei dispositivi di I/O. Solitamente, questa routine viene chiamata dopo aver iniziato una nuova comunicazione verso un dispositivo di I/O. La routine restituisce informazioni sullo stato del dispositivo, oppure gli errori che si sono verificati durante le operazioni di I/O. I bit ritornati nell'accumulatore contengono le seguenti informazioni:

POSIZIONE DEL BIT DI STATO	VALORE NUMERICO DELLO STATO	LETTURA DA REGISTRATORE	R/W SERIALE	VERIFICA+CARICO DEL NASTRO
0	1		Supero Tempo scrittura	
1	2		Supero Tempo lettura	
2	4	Blocco Corto		Blocco Corto
3	8	Blocco Lungo		Blocco Lungo
4	16	Errore lettura irrecuperabile		Qualsiasi Errore
5	32	Errore controllo sommatoria		Errore controllo sommatoria
6	64	Fine File	Fine Linea	
7	-128	Fine Nastro	Dispositivo non presente	Fine Nastro

Come si usa:

1) Chiamare questa routine.
2) Decodificare l'informazione contenuta nel registro .A quando essa si riferisce al programma.

ESEMPIO:

```
;CONTROLLA LA FINE DEL FILE DURANTE UNA LETTURA
 JSR READST
 AND #64        ;CONTROLLA IL BIT DI FINE FILE
 BNE EOF        ;SALTA SE E' FINE FILE
```

B.23 – Nome della funzione: RESTOR

Scopo: Ripristina il sistema standard ed i vettori di interruzione
Indirizzo di chiamata: $FF8A (HEX), 65418 (decimale)
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: 2
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine ripristina i valori standard di tutti i vettori di sistema usati dalle routine e dalle interruzioni del BASIC e del KERNAL (vedere la mappa di memoria per il contenuto di default dei vettori). La routine VECTOR del KERNAL e' usata per leggere e modificare individualmente i vettori di sistema.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR RESTOR
```

B.24 – Nome della funzione: SAVE

Scopo: Salva la memoria su un dispositivo
Indirizzo di chiamata: $FFD8 (HEX), 65496 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: SETLFS, SETNAM
Errori di ritorno: 5,8,9,READST
Richiesta dello stack: Nessuna
Registri interessati: .A, .X, .Y

Descrizione: – Questa routine salva un segmento di memoria a partire da un indirizzo indiretto, contenuto sulla pagina zero e specificato dall'accumulatore, fino all'indirizzo memorizzato nei registri .X e .Y; successivamente, viene trasferita ad un file logico su un dispositivo di input/output. Prima di usare questa routine, devono essere richiamate le routine SETLFS e SETNAM. Da notare che un salvataggio sul dispositivo 1 (registratore Datassette [TM]) non richiede necessariamente il nome del file. Qualsiasi altro tentativo di salvataggio su altri dispositivi senza usare il nome del file genera un errore.

NOTA: I dispositivi 0 (tastiera), 2 (RS-232) e 3 (schermo) non possono essere salvati.
Qualunque tentativo in questo senso genera un errore e l'interruzione di questa routine.

Come si usa:

0) Usare la routine SETLFS e SETNAM (a meno che non si desideri fare un salvataggio senza alcun nome di file su un registratore).
1) Caricare due locazioni consecutive su pagina zero con un puntatore all'inizio del salvataggio.
2) Caricare l'accumulatore con l'"offset" del singolo byte di pagina zero per il puntatore.
3) Caricare i registri .X e .Y rispettivamente con i byte basso ed alto della locazione dove termina il salvataggiio.
4) Chiamare questa routine.

ESEMPIO:

```
LDA #1          ;DISPOSITIVO 1 = REGISTRATORE
JSR SETLFS
LDA #0          ;NESSUN NOME DI FILE
JSR SETNAM
LDA PROG        ;CARICA L'INDIRIZZO DI PARTENZA DEL SALVATAGGIO
STA TXTTAB      ;BYTE BASSO
LDA PROG+1
STA TXTTAB+1    ;BYTE ALTO
LDX VARTAB      ;CARICA .X CON IL BYTE BASSO DI FINE SALVATAGGIO
LDY VARTAB+1    ;CARICA .Y CON IL BYTE ALTO
LDA #<TXTTAB
JSR SAVE
```

B.25 – Nome della funzione: SCNKEY

Scopo: Scandisce la tastiera
Indirizzo di chiamata: $FF9F (HEX), 65439 (decimale)

Registri di comunicazione: Nessuno
Procedura di preparazione: IOINIT
Errori di ritorno: Nessuno
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine esegue la scansione della tastiera del COMMODORE 64 individuando i tasti premuti. Questa routine viene chiamata anche dal gestore delle interruzioni. Se si preme un tasto, il corrispondente valore ASCII viene riportato nella coda della tastiera. Questa routine viene chiamata solamente se viene superata l'interruzione IRQ normale.

Come si usa:

0) Chiamare questa routine.

ESEMPIO:

```
GET JSR SCNKEY    ;SCANDISCE LA TASTIERA
    JSR GETIN     ;PRELEVA UN CARATTERE
    CMP #0        ;E' IL CARATTERE NULLO ("") ?
    BEQ GET       ;SE SI, RICOMINCIA LA SCANSIONE
    JSR CHROUT    ;ALTRIMENTI, STAMPA IL CARATTERE
```

B.26 - Nome della funzione: SCREEN

Scopo: Ritorna il formato dello schermo
Indirizzo di chiamata: $FFED (HEX), 65517 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Richiesta dello stack: 2
Registri interessati: .X, .Y

Descrizione: - Questa routine ritorna il formato dello schermo, cioe' il numero delle colonne (40) e' contenuto nel registro .X, mentre il numero delle righe (25) e' contenuto nel registro .Y. Questa routine puo' essere usata per determinare su quale tipo di macchina il programma sta funzionando. Questa funzione e' stata implementata sul COMMODORE 64 per facilitare la compatibilita' dei programmi.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR SCREEN
STX MAXCOL
STY MAXROW
```

B.27 - Nome della funzione: SECOND

Scopo: Invia l'indirizzo secondario per LISTEN
Indirizzo di chiamata: $FF93 (HEX), 65427 (decimale)
Registri di comunicazione: .A

Procedure di preparazione: LISTEN
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine e' usata per inviare un indirizzo secondario ad un dispositivo di I/O dopo che e' stata effettuata la chiamata alla routine LISTEN, predisponendo il dispositivo per la ricezione. Questa routine non puo' essere usata per inviare un indirizzo secondario dopo che si e' effettuata una chiamata alla routine TALK.
L'indirizzo secondario si usa di solito per inviare ad un dispositivo informazioni di preparazione prima di iniziare le operazioni di I/O.

Come si usa:

1) Caricare l'accumulatore con l'indirizzo secondario da inviare.
2) Chiamare la routine.

ESEMPIO:

```
;INDIRIZZA IL DISPOSITIVO #8 CON IL COMANDO (INDIRIZZO SECONDARIO)#15
    LDA #8
    JSR LISTEN
    LDA #15
    JSR SECOND
```

B.28 - Nome della funzione: SETLFS

Scopo: Predispone un file logico
Indirizzo di chiamata: $FFBA (HEX), 65466 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: Nessuno

Descrizione: - Questa routine imposta il numero del file logico, l'indirizzo del dispositivo e l'indirizzo secondario (numero del comando) per le routine del KERNAL. Il numero di file logico e' usato dal sistema come chiave di accesso alla tabella dei files creata dalla routine OPEN di apertura dei files. I valori degli indirizzi del dispositivo va da 0 a 31. I seguenti codici sono utilizzati dal COMMODORE 64 per supportare i dispositivi CBM riportati qui di seguito:

INDIRIZZO	DISPOSITIVO
0	Tastiera
1	Datassette [TM] #1
2	Dispositivo RS-232C
3	CRT Video
4	Stampante a bus seriale
8	Unita' disk CBM a bus seriale

I dispositivi di numero maggiore o uguale a 4 fanno automaticamente riferimento a dispositivi sul bus seriale. Un comando ad un dispositivo viene inviato come indirizzo secondario sul bus seriale dopo l'invio del numero del dispositivo stesso, durante la sequenza seriale che si occupa dell'"handshacking". Se non viene inviato alcun indirizzo secondario, allora il registro indice .Y deve essere impostato a 255.

Come si usa:

1) Caricare l'accumulatore con il numero di file logico.
2) Caricare il registro indice .X con il numero del dispositivo.
3) Caricare il registro indice .Y con il comando.

ESEMPIO:

PER IL FILE LOGICO 32, DISPOSITIVO #4, E NESSUN COMANDO:

```
LDA #32
LDX #4
LDY #255
JSR SETLFS
```

B.29 – Nome della funzione: SETMSG

Scopo: Controlla i messaggi di output del sistema
Indirizzo di chiamata: $FF90 (HEX), 65424 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta di stack: 2
Registri interessati: .A

Descrizione: – Questa routine controlla la stampa degli errori ed i messaggi per mezzo del KERNAL. Impostando l'accumulatore al momento della chiamata della routine, si puo' scegliere di stampare i messaggi di errore oppure i messaggi di controllo. FILE NOT FOUND e' un esempio di messaggio di errore, mentre PRESS PLAY ON CASSETTE e' un esempio di messaggio di controllo.
I bit 6 e 7 di questo valore determinano la provenienza del messaggio: se il bit 7 contiene 1, viene stampato uno dei messaggi di errore provenienti dal KERNAL; se invece viene impostato il bit 6, vengono stampati i messaggi di controllo.

Come si usa:

1) Impostare l'accumulatore al valore desiderato.
2) Chiamare questa routine.

ESEMPIO:

```
LDA #$40
JSR SETMSG      ;ATTIVA I MESSAGGI DI CONTROLLO
LDA #$80
JSR SETMSG      ;ATTIVA I MESSAGGI DI ERRORE
LDA #0
JSR SETMSG      ;DISATTIVA TUTTI I MESSAGGI DEL KERNAL
```

B.30 – Nome della funzione: SETNAM

Scopo: Predispone il nome del file
Indirizzo di chiamata: $FFBD (HEX), 65469 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedura di preparazione: Nessuna
Richiesta dello stack: Nessuna
Registri interessati: Nessuno

Descrizione: – Questa routine e' usata per predisporre il nome del file per le routine OPEN, SAVE e LOAD. L'accumulatore deve essere caricato con la lunghezza del nome del file. I registri .X e .Y devono essere caricati con l'indirizzo del nome del file nel formato standard del 6502 byte basso/byte alto. L'indirizzo puo' essere un qualsiasi indirizzo di memoria valido nel sistema dove e' memorizzata la stringa di caratteri usata per il nome del file. Se non si vuole dare un nome al file, occorre impostare nell'accumulatore il valore 0, che rappresenta una lunghezza di file pari a zero. In questo caso i registri .X e .Y possono essere impostati a qualsiasi indirizzo di memoria.

Come si usa:

1) Caricare l'accumulatore con la lunghezza del nome del file.
2) Caricare il registro indice .X con l'indirizzo piu' basso del nome del file.
3) Caricare il registro indice .Y con l'indirizzo piu' alto del nome del file.
4) Chiamare questa routine.

ESEMPIO:

```
LDA #NAME2-NAME     ;CARICA LA LUNGHEZZA DEL NOME DEL FILE
LDX #<NAME          ;CARICA L'INDIRIZZO DEL NOME DEL FILE
LDY #>NAME
JSR SETNAM
```

B.31 – Nome della funzione: SETTIM

Scopo: Imposta il clock di sistema
Indirizzo di chiamata: $FFDB (HEX), 65499 (decimale)
Registri di comunicazione: .A, .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: 2
Registri interessati: Nessuno

Descrizione: – Il clock di sistema viene gestito da una routine di interruzione che aggiorna il clock ogni 1/60 di secondo (questa quantita' viene anche indicata con "jiffy"). Il clock e' lungo tre byte, che gli consentono di contare fino a 5.184.000 "jiffy", dopodiche' ricomincia da zero. Prima di chiamare questa routine, l'accumulatore deve contenere il byte piu' significativo, il registro .X il byte piu' significativo seguente ed il registro .Y l'ultimo byte piu' significativo dell'impostazione iniziale del tempo (espresso in

"jiffy").

Come si usa:

1) Caricare l'accumulatore con l'MSB del numero di tre byte per impostare il clock.
2) Caricare il registro .X con il byte successivo.
3) Caricare il registro .Y con l'LSB (Byte meno significativo).
4) Chiamare questa routine.

ESEMPIO:

```
;IMPOSTA IL CLOCK A 10 MINUTI = 3600 "JIFFY"
 LDA #0              ;BYTE PIU' SIGNIFICATIVO
 LDX #>3600
 LDY #<3600          ;BYTE MENO SIGNIFICATIVO
 JSR SETTIM
```

B.32 – Nome della funzione: SETTMO

Scopo: Imposta l'indicatore di supero tempo della scheda del bus IEEE
Indirizzo di chiamata: $FFA2 (HEX), 65442 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richieste dello stack: 2
Registri interessati: Nessuno

NOTA: Questa routine e' usata solo con una scheda aggiuntiva IEEE.

Descrizione: – Questa routine imposta l'indicatore di supero tempo per il bus IEEE. Quando l'indicatore di supero tempo e' impostato, il COMMODORE 64 attende per 64 millisecondi la comparsa di un dispositivo sulla porta IEEE. Se tale dispositivo non risponde al segnale di Indirizzo Dati Valido (DAV – Data Address Valid) del COMMODORE 64 entro tale tempo, il COMMODORE 64 emette una condizione di errore lasciando la sequenza di "handshacking". Quando si chiama questa routine con il bit 7 dell'accumulatore impostato a 0, viene attivato il supero tempo. Se invece questo bit e' a 1, il supero tempo viene disabilitato.

NOTA: Il COMMODORE 64 usa la caratteristica del supero tempo per comunicare che non ha trovato un file su disco, durante un tentativo di apertura di file usando una scheda IEEE.

Come si usa:

PER IMPOSTARE L'INDICATORE SUPERO TEMPO:
1) Impostare a 0 il bit 7 dell'accumulatore.
2) Chiamare questa routine.

PER AZZERARE L'INDICATORE DI SUPERO TEMPO:
1) Impostare a 0 il bit 7 dell'accumulatore.
2) Chiamare questa routine.

ESEMPIO:

```
;DISABILITA IL SUPERO TEMPO
LDA #0
JSR SETTMO
```

B.33 – Nome della funzione: STOP

Scopo: Controlla se e' stato premuto il tasto STOP
Indirizzo di chiamata: $FFE1 (HEX), 65505 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richiesta dello stack: Nessuna
Registri interessati: .A, .X

Descrizione: - Se si preme il tasto STOP sulla tastiera durante una chiamata alla routine UDTIM, viene impostato l'indicatore Z. Inoltre, i canali vengono ripristinati ai valori standard. Tutti gli altri indicatori rimangono immutati. Se il tasto STOP non e' stato premuto, allora l'accumulatore contiene un byte che rappresenta l'ultima linea di scansione della tastiera. Questa routine puo' essere usata dall'Utente per controllare anche gli altri tasti.

Come si usa:

0) Chiamare la funzione UDTIM.
1) Chiamare questa routine.
2) Controllare se l'indicatore e' a zero.

ESEMPIO:

```
JSR UDTIM     ;SCANDISCE LA TASTIERA PER CONTROLLARE SE SI E' PREMUTO
              ;IL TASTO STOP
JSR STOP
BNE *+5       ;TASTO NON PREMUTO
JMP READY     ;STOP
```

B.34 – Nome della funzione: TALK

Scopo: Ordina ad un dispositivo sul bus seriale di comunicare
Indirizzo di chiamata: $FFB4 (HEX), 65460 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Per usare questa routine e' necessario innanzitutto caricare nell'accumulatore il numero del dispositivo (che deve essere compreso fra 0 e 31). Una volta chiamata, questa routine esegue una OR bit per bit per convertire il numero del dispositivo in un indirizzo di colloquio. Successivamente, questo dato viene trasmesso come

comando sul bus seriale.

Come si usa:

1) Caricare l'accumulatore con il numero del dispsitivo
2) Chiamare questa routine.

ESEMPIO:

```
;ORDINA AL DISPOSITIVO #4 DI COMUNICARE
 LDA #4
 JSR TALK
```

B.35 – Nome della funzione: TKSA

Scopo: Invia un indirizzo secondario al dispositivo predisposto al colloquio
Indirizzo di chiamata: $FF96 (HEX), 65430 (decimale)
Registri di comunicazione: .A
Procedure di preparazione: TALK
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: – Questa routine trasmette un indirizzo secondario sul bus seriale al dispositivo che deve colloquiare. Per chiamare questa routine, l'accumulatore deve contenere un numero compreso fra 0 e 31. La routine invia questo numero sul bus seriale sottoforma di comando relativo all'indirizzo secondario. Questa routine puo' essere chiamata solamente dopo aver chiamato la funzione TALK; chiamata invece dopo la funzione LISTEN, non avra' effetto.

Come si usa:

0) Chiamare la routine TALK.
1) Caricare l'accumulatore con l'indirizzo secondario.
2) Chiamare questa routine.

ESEMPIO:

```
;COMUNICA AL DISPOSITIVO #4 DI COLLOQUIARE CON IL COMANDO #7
 LDA #4
 JSR TALK
 LDA #7
 JSR TALKSA
```

B.36 – Nome della funzione: UDTIM

Scopo: Aggiorna il clock di sistema
Indirizzo di chiamata: $FFEA (HEX), 65514 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno

Richiesta di stack: 2
Registri interessati: .A, .X

Descrizione: - Questa routine aggiorna il clock di sistema. Di solito, questa routine viene chiamata da una normale routine di interruzione del KERNAL ogni 1/60 di secondo. Se il programma utente elabora interruzioni proprie, per aggiornare il tempo e' necessario chiamare questa routine. Se, inoltre, si desidera che il tasto STOP rimanga funzionale, e' necessario chiamare la routine STOP.

Come si usa:

1) Chiamare questa routine

ESEMPIO:

```
JSR UDTIM
```

B.37 - Nome della funzione: UNLSN

Scopo: Invia un comando di non-ascolto
Indirizzo di chiamata: $FFAE (HEX), 65454 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna
Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine ordina a tutti i dispositivi sul bus seriale di interrompere la ricezione di dati dal COMMODORE 64 (UNLSN = UNLISTEN - Non-ascolto). Una chiamata a questa routine provoca la trasmissione sul bus seriale di un comando UNLISTEN (non-ascolto). Questo comando interessa solamente i dispositivi precedentemente individuati. Normalmente, questa routine viene usata dopo che il COMMODORE 64 ha terminato l'invio di dati ad un dispositivo esterno. Il comando UNLISTEN, inviato a dispositivi posti in ascolto, permette di lasciare il bus seriale in modo tale che quest'ultimo possa essere impiegato per altri scopi.

Come si usa:

1) Chiamare questa routine.

ESEMPIO:

```
JSR UNLSN
```

B.38 - Nome della funzione: UNTLK

Scopo: Invia un comando di non-colloquio
Indirizzo di chiamata: $FFAB (HEX), 65451 (decimale)
Registri di comunicazione: Nessuno
Procedure di preparazione: Nessuna

Errori di ritorno: Vedere READST
Richiesta dello stack: 8
Registri interessati: .A

Descrizione: - Questa routine trasmette sul bus seriale un comando di interruzione-trasmissione. Tutti i dispositivi precedentemente disposti al colloquio interrompono l'invio di dati dal momento della ricezione di questo comando.

Come si usa:

1) Chiamare questa routine.

ESEMPIO:

```
JSR UNTALK
```

B.39 – Nome della funzione: VECTOR

Scopo: Gestisce i vettori della RAM
Indirizzo di chiamata: $FF8D (HEX), 65421 (decimale)
Registri di comunicazione: .X, .Y
Procedure di preparazione: Nessuna
Errori di ritorno: Nessuno
Richieste dello stack: 2
Registri interessati: .A, .X, .Y

Descrizione: - Questa routine gestisce tutti gli indirizzi di salto del vettore di sistema memorizzato nella RAM. Se si chiama questa routine quando il bit di riporto dell'accumulatore e' impostato, si ottiene una lista, puntata dai registri .X e .Y, comprendente la memorizzazione del contenuto attuale dei vettori della RAM. Se questa routine viene chiamata quando il bit di riporto e' azzerato, allora la lista utente puntata dai registri .X e .Y viene trasferita nei vettori della RAM di sistema.

NOTA: L'uso di questa routine richiede una certa attenzione. Il modo migliore di usare questa routine consiste nel leggere innanzitutto il contenuto del vettore nell'area utente; poi, modificare i vettori desiderati; quindi, copiare di nuovo il contenuto nei vettori di sistema.

Come si usa:

PER LEGGERE I VETTORI DELLA RAM DI SISTEMA:
1) Impostare il riporto.
2) Impostare i registri .X e .Y all'indirizzo in cui posizionare i vettori.
3) Chiamare questa routine.

PER CARICARE I VETTORI DELLA RAM DI SISTEMA:
1) Impostare il riporto.
2) Impostare i registri .X e .Y all'indirizzo della lista dei vettori della RAM da caricare
3) Chiamare questa routine.

ESEMPIO:

```
;CARICA LE ROUTINE DI INPUT NEL NUOVO SISTEMA
LDX #<USER
LDY #>USER
SEC
JSR VECTOR          ;LEGGE I VECCHI VETTORI
LDA #<MYINP         ;CAMBIA L'INPUT
STA USER+11
LDX #<USER
LDY #>USER
CLC
JSR VECTOR          ;CAMBIA IL SISTEMA
...
USER *=*+26
```

CODICI ERRORE

La seguente lista riporta i **messaggi** di errore che possono verificarsi usando le routine del KERNAL. Se si verifica un errore durante lo svolgimento di una routine del KERNAL, viene impostato il bit di riporto dell'accumulatore, e nel medesimo viene riportato il numero corrispondente al messaggio di errore.

NOTA: Alcune routine di I/O del KERNAL non usano questi codici di errore. In questo caso gli errori possono essere identificati usando la routine READST del KERNAL.

CODICE	SIGNIFICATO
0	Routine terminata dal tasto STOP
1	Troppi file aperti
2	File gia' aperto
3	File non aperto
4	File non trovato
5	Dispositivo non presente
6	File non di input
7	File non di output
8	Manca il nome del file
9	Numero di dispositivo illegale
240	La cima della memoria modifica l'allocazione/disallocazione del buffer RS-232

USO DEL LINGUAGGIO MACCHINA DA BASIC

Esistono numerosi modi per usare il BASIC ed il linguaggio macchina sul COMMODORE 64, comprese particolari istruzioni come parte del CBM BASIC, come pure locazioni chiave nella macchina. Ci sono cinque modi principali di usare sul COMMODORE 64 le routine in linguaggio macchina:

1) L'istruzione BASIC SYS
2) La funzione BASIC USR
3) La modifica di uno dei vettori di I/O della RAM
4) La modifica di uno dei vettori di interruzione della RAM
5) La modifica della routine CHRGET

1) L'istruzione BASIC SYS X provoca il salto ad una sottoprocedura in linguaggio macchina allocata nell'indirizzo X. Questa routine deve terminare con un'istruzione RTS (ReTurn from Subroutine - RiTorno da Sottoprocedura), che ritorna il controllo al BASIC. Tra la routine in linguaggio macchina ed il programma in BASIC, i parametri vengono passati usando le istruzioni BASIC PEEK e POKE ed i loro equivalenti in linguaggio macchina.
Il comando SYS e' il metodo piu' comune per unire il BASIC al linguaggio macchina. La PEEK e la POKE consentono un facile passaggio di parametri multipli. In un programma ci possono essere diverse istruzioni SYS, ciascuna destinata a differenti (in qualche caso, alla stessa) routine in linguaggio macchina.

2) La funzione BASIC USR(X) passa il controllo alla sottoprocedura in Linguaggio Macchina allocata nella posizione indicata dall'indirizzo memorizzato nelle locazioni 785 e 786 (tale indirizzo e' memorizzato nel formato standard byte basso/byte alto). Il valore di X viene valutato e passato alla sottoprocedura in Linguaggio Macchina tramite l'accumulatore reale #1, il cui inizio e' stabilito all'indirizzo $61 (per ulteriori dettagli si veda la mappa di memoria). Al programma BASIC puo' essere ritornato un valore, preventivamente sistemato nell'accumulatore reale; il ritorno al BASIC si verifica se la routine termina con l'istruzione RTS.
L'istruzione USR(X) e' diversa da SYS, in quanto necessita dell'impostazione di un vettore indiretto, che rappresenta il formato attraverso il quale la variabile viene passata (formato in virgola mobile). Se si usa piu' di una routine in Linguaggio Macchina, e' necessario modificare il vettore.

3) Ciascuna routine interna di input/output o del BASIC, a cui si fa accesso per mezzo della tabella dei vettori allocata a pagina 3 (vedere MODI DI INDIRIZZAMENTO, PAGINA ZERO), puo' essere sostituita o modificata da un codice utente. Ogni vettore di due byte e' formato da un indirizzo, composto da un byte alto ed un byte basso, a disposizione del sistema operativo.
La routine VECTOR del KERNAL rappresenta il modo piu' sicuro per modificare uno qualunque di tali vettori; tuttavia, un singolo vettore puo' essere modificato anche dall'istruzione POKE. In

questo caso, un nuovo vettore punta alla routine preparata dall'Utente e che ha lo scopo di sostituire o incrementare la routine standard del sistema. La routine Utente viene lanciata al momento dell'esecuzione del comando BASIC appropriato. Se dopo l'esecuzione di questa routine e' necessario eseguire una normale routine di sistema, allora il programma Utente deve prevedere un'istruzione di salto (JMP) all'indirizzo precedentemente conetnuto nel vettore. In caso contrario, la routine deve terminare con un'istruzione RTS che restituisce il controllo al BASIC.

4) Si puo' modificare il VETTORE DELLE INTERRUZIONI HARDWARE (IRQ). Ogni 1/60 di secondo il sistema operativo passa il controllo alla routine, specificata da questo vettore, che di solito viene usata per la sincronizzazione, per la scansione della tastiera, ecc. Se si usa questa tecnica occorre trasferire sempre il controllo alla routine di gestione dell'IRQ normale, a meno che la routine di sostituzione sia in grado di operare sul circuito CIA (se CIA e' gestito dalla routine, RICORDARSI di terminare la routine con l'istruzione RTI [ReTurn from Interrupt - RiTorno da Interruzione]).
   Questo metodo e' valido per la determinazione di quello che si deve verificare in concorrenza ad un programma BASIC, ma presenta lo svantaggio di essere piu' difficile.

NOTA: PRIMA DI MODIFICARE QUESTO VETTORE, DISABILITARE SEMPRE LE INTERRUZIONI !

5) La routine CHRGET e' usata dal BASIC per prelevare ogni carattere o simbolo, cosi' da semplificare l'aggiunta di nuovi comandi BASIC. Naturalmente, ciascun nuovo comando deve essere eseguito da una sottoprocedura in Linguaggio Macchina scritta dall'Utente. Un modo comune per usare questo metodo e' quello di specificare un carattere (@, per esempio) che deve comparire prima di ogni altro nuovo comando. La nuova routine CHRGET cerca i caratteri speciali: se non ne vengono trovati, il controllo passa alla normale routine CHRGET del BASIC; in caso contrario, il nuovo comando viene interpretato ed eseguito dal programma in Linguaggio Macchina. Si minimizza cosi' il tempo extra di esecuzione necessario alla ricerca dei comandi aggiuntivi. Questa tecnica viene spesso indicata come "wedge".

DOVE MEMORIZZARE LE ROUTINE IN LINGUAGGIO MACCHINA

Il posto migliore per allocare le routine in linguaggio macchina sul COMMODORE 64 sono le locazioni da $C000 a $CFFF, assumendo che le routine siano piu' corte di 4K byte. Questo segmento di memoria non e' influenzato dal BASIC.

Se non e' possibile, o non si desidera, inserire le routine in linguaggio macchina a partire da $C000, per esempio perche' sono piu' lunghe di 4K byte, allora e' necessario proteggere dal BASIC un'area di memoria a partire dalla cima della memoria. Quest'ultima e' allocata in $9FFF, e puo' essere modificata usufruendo della routine MEMTOP del KERNAL, oppure della seguente istruzione:

```
10 POKE51,L:POKE52,H:POKE55,L:POKE56,H:CLR
```

dove H e L sono rispettivamente la porzione alta e bassa della nuova cima della memoria. Ad esempio, per riservare al Linguaggio Macchina l'area compresa fra $9000 e $9FFF, si puo' usare la seguente istruzione:

```
10 POKE51,0:POKE52,144:POKE55,0:POKE56,144:CLR
```

COME SI ACCEDE AL LINGUAGGIO MACCHINA

Per aggiungere programmi in linguaggio macchina ad un programma BASIC ci sono tre modi comuni:

1) ISTRUZIONI DATA:

Le routine in linguaggio macchina possono essere implementate dalla lettura di istruzioni DATA e dall'inserimento di valori (tramite l'istruzione POKE) in memoria all'inizio del programma. Questo e' il metodo piu' facile: non sono necessari metodi speciali per salvare le due parti del programma, ed e' facile da correggere. Gli inconvenienti comportano l'occupazione di maggiore spazio di memoria e l'attesa del caricamento del programma. Percio', questo metodo e' consigliabile per routine brevi.

ESEMPIO:

```
10 RESTORE:FORX=1TO9:READA:POKE12*4096+X,A:NEXT
.
.
.
BASIC PROGRAM
.
.
.
1000 DATA 161,1,204,204,204,204,204,204,96
```

2) MONITOR DEL LINGUAGGIO MACCHINA (64MON):

Questo programma permette di accedere ad un programma in entrambi i codici ESADECIMALE e SIMBOLICO, e di salvare il segmento di memoria in cui si trova il programma. I vantaggi di questo metodo sono un piu' facile accesso alla routine in linguaggio macchina, semplificazioni della messa a punto ed un buon numero di mezzi veloci di salvataggio e caricamento. L'inconveniente e' rappresentato dalla richiesta al programma BASIC di caricare all'inizio la routine in linguaggio macchina da cassetta o da disco (per ulteriori chiarimenti sulla cartuccia 64MON si veda la sezione sul linguaggio macchina).

ESEMPIO:

Il seguente e' un esempio di programma BASIC che usa una routine in Linguaggio Macchina preparata dalla cartuccia 64MON. La routine e' memorizzata su cassetta.

```
10 IF FLAG=1 THEN 20
15 FLAG=1:LOAD "MACHINE LANGUAGE ROUTINE NAME",1,1
20
  .
  .
  .
 PARTE RIMANENTE DEL PROGRAMMA BASIC
```

3) PACKAGE EDITOR/ASSEMBLER:

I vantaggi e gli svantaggi sono simili all'uso del monitor del linguaggio macchina, solo che l'accesso ai programmi e' piu' semplice.

MAPPA DELLA MEMORIA DEL COMMODORE 64

LABEL	INDIRIZZO ESADECIMALE	LOCAZIONE ESADECIMALE	DESCRIZIONE
D6510	0000	0	Registro direzione dati del circuito 6510
R6510	0001	1	Registro a 8 bit di Input/Output del circuito 6510
	0002	2	Non usato
ADRAY1	0003-0004	3-4	Vettore salti: Conversione reale-intero
ADRAY2	0005-0006	5-6	Vettore salti: Conversione intero-reale
CHARAC	0007	7	Carattere di ricerca
ENDCHR	0008	8	Indicatore: Cerca le virgolette alla fine di una stringa
TRMPOS	0009	9	Colonna di schermo dopo l'ultima TAB
VERCK	000A	10	Indicatore: 0=Carica, 1=Verifica
COUNT	000B	11	Puntatore buffer di input/numero indici
DIMFLG	000C	12	Indicatore: Dimensione di default di una schiera
VALTYP	000D	13	Tipo di dato: $FF=Stringa, $00=Numerico
INTFLG	000E	14	Tipo di dato: $80=Intero, $00=Reale
GARBFL	000F	15	Scansione istruzione DATA/Virgolette istruzione LIST/"Garbage Collection"
SUBFLG	0010	16	Indicatore: Riferimento indice/Chiamata di funzione Utente
INPFLG	0011	17	Indicatore: $00=INPUT, $40=GET, $98=READ
TANSGN	0012	18	Indicatore: Simbolo TAN/Risultato di un confronto
	0013	19	Indicatore: Richiesta di INPUT
LINNUM	0014-0015	20-21	Transiente: Valore intero
TEMPPT	0016	22	Puntatore: Stack stringhe tansienti
LASTPT	0017-0018	23-24	Ultimo indirizzo stringhe transienti
TEMPST	0019-0021	25-33	Stack stringhe transienti
INDEX	0022-0025	34-37	Area puntatori programmi di utilita'
RESHO	0026-002A	38-42	Prodotto di moltiplicazione reale
TXTTAB	002B-002C	43-44	Puntatore: Inizio del testo BASIC
VARTAB	002D-002E	45-46	Puntatore: Inizio variabili del BASIC
ARYTAB	002F-0030	47-48	Puntatore: Inizio schiere del BASIC
STREND	0031-0032	49-50	Puntatore: Fine schiere del BASIC (+1)
FRETOP	0033-0034	51-52	Puntatore: Base della memoria stringa
FRESPC	0035-0036	53-54	Puntatore stringa programmi di utilita'
MEMSIZ	0037-0038	55-56	Puntatore: Indirizzo piu' alto usato dal BASIC
CURLIN	0039-003A	57-58	Numero di linea corrente del BASIC
OLDLIN	003B-003C	59-60	Numero di linea precedente del BASIC
OLDTXT	003D-003E	61-62	Puntatore: Istruzione BASIC per CONT
DATLIN	003F-0040	63-64	Numero di linea DATA corrente
DATPTR	0041-0042	65-66	Puntatore: Indirizzo elemento corrente dell'istruzione DATA
INPPTR	0043-0044	67-68	Vettore: Routine di INPUT
VARNAM	0045-0046	68-69	Nome variabile corrente del BASIC
VARPNT	0047-0048	70-71	Puntatore: Dato variabile corrente

			del BASIC
FORPNT	0049-004A	73-74	Puntatore: Variabile indice per il ciclo FOR...NEXT
	004B-0060	75-96	Area puntatore / dati transiente
FACEXP	0061	97	Accumulatore reale #1: Esponente
FACHO	0062-0065	98-101	Accumulatore reale #1: Mantissa
FACSGN	0066	102	Accumulatore reale #1: Segno
SGNFLG	0067	103	Puntatore: Costante di valutazione delle serie
BITS	0068	104	Accumulatore reale #1: Cifra di overflow
ARGEXP	0069	105	Accumularore reale #2: Esponente
ARGHO	006A-006D	106-109	Accumulatore reale #2: Mantissa
ARGSGN	006E	110	Accumulatore reale #2: Segno
ARISGN	006F	111	Risultato di confronto del segno: Accumulatore #1 contro Accumulatore #2
FACOV	0070	112	Accumulatore reale #1. Byte basso (arrotondamento)
FBUFPT	0071-0072	113-114	Puntatore: Buffer cassetta
CHRGET	0073-008A	115-138	Sottoprocedura: Preleva il prossimo byte del testo BASIC
CHRGOT	0079	121	Ingresso per un nuovo prelievo dello stesso byte di testo
TXTPTR	007A-007B	122-123	Puntatore: Byte corrente del testo BASIC
RNDX	008B-008F	139-143	Valore reale del seme della funzione RND
STATUS	0090	144	Parola di stato dell'I/O del KERNAL: ST
STKEY	0091	145	Indicatore: Tasto STOP / Tasto RVS
SVXT	0092	146	Costante di misura del tempo per nastro
VERCK	0093	147	Indicatore: 0=Carica, 1=Verifica
C3PO	0094	148	Indicatore: Bus seriale - Carattere bufferizzato di output
BSOUR	0095	149	Carattere bufferizzato per bus seriale
SYNO	0096	150	Numero di sincronismo cassetta
	0097	151	Area dati transiente
LDTND	0098	152	Numero file aperti/Indice della tabella dei file
DFLTN	0099	153	Dispositivo di input di default (0)
DFLTO	009A	154	Dispositivo di output (CMD) di default (3)
PRTY	009B	155	Parita' carattere nastro
DPSW	009C	156	Indicatore: Ricevuto byte da nastro
MSGFLG	009D	157	Indicatore: $80=Modo diretto, $00=Modo Programma
PTR1	009E	158	Registro errore passo 1 del nastro
PTR2	009F	159	Registro errore passo 2 del nastro
TIME	00A0-00A2	160-162	Clock in tempo reale (approsimato) ad 1/60 di secondo ("Jiffy")
	00A3-00A4	163-164	Area dati transiente
CNTDN	00A5	165	Contatore a ritroso di sincronizzazione cassetta
BUFPNT	00A6	166	Puntatore: Buffer di I/O del nastro
INBIT	00A7	167	Bit di input dell'RS-232/Cassetta Transiente
BITCI	00A8	168	Contatore bit di input dell'RS-232/ Cassetta transiente

RINONE	00A9	169	Indicatore RS-232: Controllo del bit di partenza
RIDATA	00AA	170	Buffer del byte di input dell'RS-232/ Cassetta tansiente
RIPRTY	00AB	171	Parita' input dell'RS-232/Contatore corto cassetta
SAL	00AC-00AD	172-173	Puntatore: Buffer nastro/Scorrimento schermo
EAL	00AE-00AF	174-175	Indirizzi di Fine nastro/Fine programma
CMP0	00B0-00B1	176-177	Costanti di misura del tempo del nastro
TAPE1	00B2-00B3	178-179	Puntatore: Inizio buffer del nastro
BITTS	00B4	180	Contatore bit di output dell'RS-232/ Cassetta transiente
NXTBIT	00B5	181	Prossimo bit dell'RS-232 da inviare/ Indicatore di fine nastro (EOT)
RODATA	00B6	182	Buffer del byte di output dell'RS-232
FNLEN	00B7	183	Lunghezza del nome del file corrente
LA	00B8	184	Numero file logico corrente
SA	00B9	185	Indirizzo secondario corrente
FA	00BA	186	Numero del dispositivo corrente
FNADR	00BB-00BC	187-188	Puntatore: Nome del file corrente
ROPRTY	00BD	189	Parita' output dell'RS-232/Cassetta transiente
FSBLK	00BE	190	Contatore blocco Read/Write cassetta
MYCH	00BF	191	Buffer parola seriale
CAS1	00C0	192	Arresto motore del nastro
STAL	00C1-00C2	193-194	Indirizzo di partenza dell'I/O
MEMUSS	00C3-00C4	195-196	Carico nastro transiente
LSTX	00C5	197	Tasto corrente premuto: CHR$(n) 0=Nessun tasto
NDX	00C6	198	Numero caratteri nel buffer della tastiera (coda)
RVS	00C7	199	Indicatore: Stampa caratteri inversi - 1=Si, 0=Non usato
INDX	00C8	200	Puntatore: Fine linea logica per INPUT
LXSP	00C9-00CA	201-202	Posizione (X,Y) del cursore all'inizio di INPUT
SFDX	00CB	203	Indicatore: Stampa i caratteri ottenuti tenendo premuto il tasto SHIFT
BLNSW	00CC	204	Abilitatore del lampeggio: 0=Lampeggio
BLNCT	00CD	205	Timer: Conto alla rovescia per cursore bistabile
GDBLN	00CE	206	Carattere sotto il cursore
BLNON	00CF	207	Indicatore: Ultima impostazione cursore (lampeggio/fisso)
CRSW	00D0	208	Indicatore: INPUT o GET da tastiera
PNT	00D1-00D2	209-210	Puntatore: Indirizzo della linea di schermo corrente
PNTR	00D3	211	Colonna del cursore sulla linea corrente
QTSW	00D4	212	Indicatore: Editor modo "quote", $00=NO
LNMX	00D5	213	Lunghezza linea di schermo fisica
TBLX	00D6	214	Numero linea fisica attuale del cursore
	00D7	215	Area dati transiente
INSRT	00D8	216	Indicatore: Modo Inserimento >0 = # INST
LDTB1	00D9-00F2	217-242	Tavola collegamenti della linea dello

			schermo/Editor transiente
USER	00F3-00F4	243-244	Puntatore: Locazione corrente della RAM colore dello schermo
KEYTAB	00F5-00F6	245-246	Vettore: Tavola di decodificazione della tastiera
RIBUF	00F7-00F8	247-248	Puntatore al buffer di input dell'RS-232
ROBUF	00F9-00FA	249-250	Puntatore al buffer di output dell'RS-232
FREKZP	00FB-00FE	251-254	Libera Pagina 0 per programmi Utente
BASZPT	00FF	255	Area dati transiente del BASIC
	0100-01FF	256-511	Area stack sistema del microprocessore
	0100-010A	256-266	Fluttuante per area di lavoro stringa
BAD	0100-013E	256-318	Registro errori di input del nastro
BUF	0200-0258	512-600	Buffer di INPUT del sistema
LAT	0259-0262	601-610	Tabella KERNAL: Numero file logici attivi
FAT	0263-026C	611-620	Tabella KERNAL: Numero dispositivo per ogni file
SAT	026D-0276	621-630	Tabella KERNAL:Indirizzo secondario di ogni file
KEYD	0277-0280	631-640	Coda del buffer della tastiera (FIFO)
MEMSTR	0281-0282	641-642	Puntatore: Base della memoria per Sistema Operativo
MEMSIZ	0283-0284	643-644	Puntatore: Cima della memoria per Sistema Operativo
TIMOUT	0285	645	Indicatore: Variabile KERNAL per supero Tempo dell'IEEE
COLOR	0286	646	Codice colore del carattere corrente
GDCOL	0287	647	Colore di fondo sotto il cursore
HIBASE	0288	648	Cima della memoria schermo (pagina)
XMAX	0289	649	Misura del buffer della tastiera
RPTFLG	028A	650	Indicatore: Ripete il tasto battuto, $80=Ripete
KOUNT	028B	651	Ripete il contatore velocita'
DELAY	028C	652	Ripete il contatore ritardo
SHFLAG	028D	653	Indicatore: Tasto SHIFT della tastiera/ Tasto CTRL/Tasto C=
LSTSHF	028E	654	Ultima configurazione ottenuta con il tasto SHIFT della tastiera
KEYLOG	028F-0290	655-656	Vettore: Preparazione tabella tastiera
MODE	0291	657	Indicatore: $00=Disabilita tasti SHIFT, $80=Abilita tasti SHIFT
AUTODN	0292	658	Indicatore: Scorrimento automatico verso il basso, 0=ON
M51CTR	0293	659	RS-232: Immagine registro di controllo del 6551
M51CDR	0294	660	RS-232: Immagine del registro di comando del 6551
M51AJB	0295-0296	661-662	BPS RS-232 USA non standard (Tempo/2-100)
RSSTAT	0297	663	RS-232: Immagine del registro di stato del 6551
BITNUM	0298	664	Numero di bit dell'RS-232 rimasti da inviare
BAUDOFF	0299-029A	665-666	Trasmittanza dell'RS-232: Tempo per un bit completo (nsec)

RIDBE	029B	667	Indice RS-232 per termine buffer input
RIDBS	029C	668	Inizio del buffer di input dell'RS-232 (pagina)
RODBS	029D	669	Inizio del buffer di output dell'RS-232 (pagina)
RODBE	029E	670	Indice RS-232 per termine buffer output
IRQTMP	029F-02A0	671-672	Contiene il vettore IRQ durante l'I/O del nastro
ENABL	02A1	673	Abilita l'RS-232
	02A2	674	Lettura di TOD durante I/O cassetta
	02A3	675	Memorizzazione transiente per lettura cassetta
	02A4	676	Indicatore D1IRQ transiente per lettura cassetta
	02A5	677	Transiente per indice di linea
	02A6	678	Indicatore PAL/NTSC, 0=NTSC, 1=PAL
	02A7-02FF	679-767	Non usati
IERROR	0300-0301	768-769	Vettore: Stampa i messaggi di errore del BASIC
IMAIN	0302-0303	770-771	Vettore: Partenza a caldo del BASIC
ICRNCH	0304-0305	772-773	Vettore: Testo BASIC "tokenizzato"
IQPLOP	0306-0307	774-775	Vettore: Lista del testo BASIC
IGONE	0308-0309	776-777	Vettore: Invio caratteri BASIC
IEVAL	030A-030B	778-779	Vettore: Valutazione "token" del BASIC
SAREG	030C	780	Memorizzazione del registro .A del 6502
SXREG	030D	781	Memorizzazione del registro .X
SYREG	030E	782	Memorizzazione del registro .Y
SPREG	030F	783	Memorizzazione del registro .SP
USRPOK	0310	784	Istruzione di salto della funzione USR
USRADD	0311-0312	785-786	Byte basso/alto dell'indirizzo di USR
	0313	787	Non usato
CINV	0314-0315	788-789	Vettore: Interruzione hardware di IRQ
CBINV	0316-0317	790-791	Vettore: Interruzione istruzione BRK
NMINV	0318-0319	792-793	Vettore: Interruzione non mascherabile
IOPEN	031A-031B	794-795	Vettore routine OPEN del KERNAL
ICLOSE	031C-031D	796-797	Vettore routine CLOSE del KERNAL
ICHKIN	031E-031F	798-799	Vettore routine CHKIN del KERNAL
ICKOUT	0320-0321	800-801	Vettore routine CHKOUT del KERNAL
ICLRCH	0322-0323	802-803	Vettore routine CLRCHN del KERNAL
IBASIN	0324-0325	804-805	Vettore routine CHRIN del KERNAL
IBSOUT	0326-0327	806-807	Vettore routine CHROUT del KERNAL
ISTOP	0328-0329	808-809	Vettore routine STOP del KERNAL
IGETIN	032A-032B	810-811	Vettore routine GETIN del KERNAL
ICLALL	032C-032D	812-813	Vettore routine CLALL del KERNAL
USRCMD	032E-032F	814-815	Vettore definito dall'Utente
ILOAD	0330-0331	816-817	Vettore routine LOAD del KERNAL
ISAVE	0332-0333	818-819	Vettore routine SAVE del KERNAL
	0334-033B	820-827	Non usati
TBUFFR	033C-03FB	828-1019	Buffer di I/O del nastro
	03FC-03FF	1020-1023	Non usati
VICSCN	0400-07FF	1024-2047	Area memoria schermo (1024 byte)
	0400-07E7	1024-2023	Matrice video (25 linee X 40 colonne)
	07F8-07FF	2040-2047	Puntatori ai dati animazione
	0800-9FFF	2048-40959	Spazio normale dei programmi BASIC
	8000-9FFF	32768-40959	ROM cartuccia VSP (8192 byte)
	A000-BFFF	40960-49151	ROM BASIC (8192 byte - 8K RAM)
	C000-CFFF	49152-53247	RAM (4096 byte)

D000-DFFF	53248-57343	Dispositivi di I/O e RAM colore, oppure ROM generatore caratteri, oppure RAM (4096 byte)
E000-EFFF	57344-65535	ROM del KERNAL (8192 byte oppure 8K RAM)

ASSEGNAZIONI DI INPUT/OUTPUT DEL COMMODORE 64

ESADECIMALE	DECIMALE	BIT	DESCRIZIONE
0000	0	7-0	Registro direzione dati del 6510 MOS (xx101111) Bit=1:Output, Bit=0:Input, x=Non usato
0001	1		Porta I/O del circuito microprocessore 6510 MOS
		0	Segnale /LORAM(0=Disattiva ROM BASIC)
		1	Segnale /HIRAM(0=Disattiva ROM KERNAL)
		2	Segnale /CHAREN(0=Attiva ROM carattere)
		3	Linea dati di output della cassetta
		4	Lettura interruttore cassetta 1 = Interruttore chiuso
		5	Controllo motore cassetta: 0=ON, 1=OFF
		6-7	Non definiti
D000-D02E	53248-54271		CONTROLLORE INTERFACCIA VIDEO (VIC) MOS 6566
D000	53248		Posizione X dell'animazione 0
D001	53249		Posizione Y dell'animazione 0
D002	53250		Posizione X dell'animazione 1
D003	53251		Posizione Y dell'animazione 1
D004	53252		Posizione X dell'animazione 2
D005	53253		Posizione Y dell'animazione 2
D006	53254		Posizione X dell'animazione 3
D007	53255		Posizione Y dell'animazione 3
D008	53256		Posizione X dell'animazione 4
D009	53257		Posizione Y dell'animazione 4
D00A	53258		Posizione X dell'animazione 5
D00B	53259		Posizione Y dell'animazione 5
D00C	53260		Posizione X dell'animazione 6
D00D	53261		Posizione Y dell'animazione 6
D00E	53262		Posizione X dell'animazione 7
D00F	53263		Posizione Y dell'animazione 7
D010	53264		Posizione X delle animazioni 0-7 (MSB delle coordinate X)
D011	53265		Registro di controllo del VIC
		7	Comparatore di quadro (Bit 8): Vedere 53266
		6	Modo Testo colore esteso 1=Abilitato
		5	Modo Bit Map - 1=Abilitato
		4	Riempie lo schermo con il colore del bordo - 0=Vuoto
		3	Seleziona le righe di testo (24 o 25) video 1=25 Righe
		2-0	Scorrimento rallentato fino alla posizione Y di un punto (0-7)
D012	53266		Lettura/Scrittura del valore di quadro per confronto con IRQ
D013	53267		Posizione X del "latch" penna ottica
D014	53268		Posizione Y del "latch" penna ottica

D015	53269		Abilitatore animazione di schermo 1=Abilitato
D016	53270		Registri di controllo del VIC
		7-6	Non usati
		5	QUESTO BIT DEVE ESSERE SEMPRE 0
		4	Modo Multicolore - 1=Abilitato (Testo o Bit Map)
		3	Seleziona le colonne sdel testo video 1=40 Colonne
		2-0	Scorrimento rallentato a posizione X
D017	53271		Espansione (2X) verticale (Y) animazioni 0-7
D018	53272		Registro di controllo della memoria del VIC
		7-4	Indirizzo base della matrice video (interno al VIC)
		3-1	Indirizzo base del punto di un carattere (interno al VIC)
D019	53273		Registro indicatore di interruzione (Bit=1: si e' verificata una IRQ)
		7	Imposta a ON qualunque condizione IRQ abilitata del VIC
		3	Indicatore IRQ triggerato per penna ottica
		2	Indicatore IRQ di contatto fra due animazioni
		1	Indicatore IRQ di contatto animazione-fondo
		0	Indicatore IRQ di comparazione quadro
D01A	53274		Registro maschera IRQ - 1=Interruzione abilitata
D01B	53275		Priorita' di schermo animazione-fondo 1=Animazione
D01C	53276		Seleziona il Modo Multicolore per le animazioni 0-7 - 1=Modo Multicolore
D01D	53277		Espansione (2X) orizzontale (X) delle animazioni 0-7
D01E	53278		Scoperta contatto fra due animazioni
D01F	53279		Scoperta di contatto animazione-fondo
D020	53280		Colore del bordo
D021	53281		Colore di fondo 0
D022	53282		Colore di fondo 1
D023	53283		Colore di fondo 2
D024	53284		Colore di fondo 3
D025	53285		Registro 0 animazione multicolore
D026	53286		Registro 1 animazione multicolore
D027	53287		Colore animazione 0
D028	53288		Colore animazione 1
D029	53289		Colore animazione 2
D02A	53290		Colore animazione 3
D02B	53291		Colore animazione 4
D02C	53292		Colore animazione 5
D02D	53293		Colore animazione 6
D02E	53294		Colore animazione 7
D400-D7FF	54272-55295		DISPOSITIVO MOS 6581 INTERFACCIA DEL SUONO (SID)
D400	54272		Voce 1: Controllo frequenza

			Byte basso
D401	54273		Voce 1: Controllo frequenza Byte alto
D402	54274		Voce 1: Ampiezza forma d'onda Pulsazione - Byte basso
D403	54275	7-4	Non usati
		3-0	Voce 1: Ampiezza forma d'onda Pulsazione - Semibyte alto
D404	54276		Voce 1: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda triangolare 1=ON
		3	Bit di controllo: 1=Disabilita l'Oscillatore 1
		2	Modulazione ad anello Oscillatore 1 con output Oscillatore 3 - 1=ON
		1	Sincronizza Oscillatore 1 con frequenza Oscillatore 3 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ DECADERE/SOSTENERE, 0=Attiva RILASCIO
D405	54277		Generatore 1 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona la durata del ciclo ATTACCARE: 0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D406	54278		Generatore 1 dell'inviluppo: ciclo di controllo SOSTENERE/RILASCIARE
		7-4	Seleziona la durata del ciclo SOSTENERE: 0-15
		3-0	Seleziona la durata del ciclo RILASCIARE: 0-15
D407	54279		Voce 2: Controllo frequenza Byte basso
D408	54280		Voce 2: Controllo frequenza Byte alto
D409	54281		Voce 2: Ampiezza forma d'onda pulsazione - Byte basso
D40A	54282	7-4	Non usati
		3-0	Voce 2: Ampiezza forma d'onda Pulsazione - Semibyte alto
D40B	54283		Voce 2: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda Triangolare 1=ON
		3	Bit di controllo: 1=Disabilita Oscillatore 2
		2	Modulazione ad anello Oscillatore 2

			con l'uscita Oscillatore 1 - 1=ON
		1	Sincronizza Oscillatore 2 con frequenza Oscillatore 1 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ SOSTENERE/DECADERE, 0=Attiva RILASCIO
D40C	54284		Generatore 2 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona durata ciclo ATTACCARE:0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D40D	54285		Generatore 2 dell'inviluppo: ciclo di controllo SOSTENERE/RILASCIARE
		7-4	Seleziona la durata del ciclo SOSTENERE: 0-15
		3-0	Seleziona la durata del ciclo RILASCIARE: 0-15
D40E	54286		Voce 3: Controllo frequenza Byte basso
D40F	54287		Voce 3: Controllo frequenza-Byte alto
D410	54288		Voce 3: Ampiezza forma d'onda PULSAZIONE - Byte basso
D411	54289	7-4	Non usati
		3-0	Voce 3: Ampiezza forma d'onda PULSAZIONE - Semibyte alto
D412	54290		Voce 3: Registri di controllo
		7	Seleziona forma d'onda Rumore Casuale 1=ON
		6	Seleziona forma d'onda Pulsazione 1=ON
		5	Seleziona la forma d'onda "dente di sega" - 1=ON
		4	Seleziona forma d'onda Triangolare 1=ON
		3	Bit di controllo: 1=Disabilita l'Oscillatore 3
		2	Modulazione ad anello Oscillatore 3 con uscita Oscillatore 2 - 1=ON
		1	Sincronizza Oscillatore 3 con frequenza Oscillatore 2 - 1=ON
		0	Bit di porta: 1=Attiva ATTACCARE/ DECADERE/SOSTENERE 0=Attiva RILASCIARE
D413	54291		Generatore 3 dell'inviluppo: ciclo di controllo ATTACCARE/DECADERE
		7-4	Seleziona durata ciclo ATTACCARE 0-15
		3-0	Seleziona durata ciclo DECADERE: 0-15
D414	54292		Generatore 3 dell'inviluppo: ciclo di controllo DECADERE/RILASCIARE
		7-4	Seleziona durata ciclo SOSTENERE:0-15
		3-0	Seleziona durata ciclo RILASCIARE: 0-15
D415	54293		Frequenza di taglio del filtro: semibyte basso (bit 2-0)
D416	54294		Frequenza di taglio del filtro: semibyte alto
D417	54295		Controllo risonanza filtro/Controllo ingresso voce

		7-4	Seleziona risonanza del filtro: 0-15
		3	Ingresso esterno filtro: 1=Si, 0=No
		2	Uscita Voce 3 del filtro: 1=Si, 0=No
		1	Uscita Voce 2 del filtro: 1=Si, 0=No
		0	Uscita Voce 1 del filtro: 1=Si, 0=No
D418	54296		Seleziona modo e volume del filtro
		7	Uscita taglio della Voce 3: 1=Si, 0=No
		6	Seleziona modo passa alto del filtro 1=ON
		5	Seleziona modo passa banda del filtro 1=ON
		4	Seleziona modo passa basso del filtro 1=ON
		3-0	Seleziona volume di uscita: 0-15
D419	54297		Convertitore analogico/digitale: Paddle 1 (0-255)
D41A	54298		Convertitore analogico/digitale: Paddle 2 (0-255)
D41B	54299		Generatore numeri casuali Oscillatore 3
D41C	54300		Uscita generatore 3 dell'inviluppo
D500-D7FF	54528-55295		IMMAGINI DEL SID
D800-DBFF	55296-56319		RAM colore (Semibytes)
DC00-DCFF	56320-56575		Adattatore Interfaccia Complessa (CIA) #1 del 6526 MOS
DC00	56320		Porta dati A (tastiera, joystick, paddle, penna ottica)
		7-0	Scrive i valori della colonna della tastiera per la sua scansione
		7-6	Legge le Paddle sulla porta A/B (01=Porta A, 10=Porta B)
		4	Pulsante sparo joystick A - 1=Fuoco
		3-2	Pulsanti sparo Paddle
		3-0	Direzione joystick A (0-15)
DC01	56321		Porta dati B (tastiera, joystick, paddle)
		7-0	Legge i valori della colonna della tastieraper la sua scansione
		7	Timer B: Uscita bistabile/pulsazione
		6	Timer A: Uscita bistabile/pulsazione
		4	Pulsante sparo joystick 1 - 1=Fuoco
		3-2	Pulsanti sparo paddle
		3-0	Direzione joystick 1
DC02	56322		Registro direzione dati Porta A(56320)
DC03	56323		Registro direzione dati Porta B(56321)
DC04	56324		Timer A: Byte basso
DC05	56325		Timer A: Byte alto
DC06	56326		Timer B: Byte basso
DC07	56327		Timer B: Byte alto

DC08	56328		Clock tempo-del-giorno: Decimi di secondo
DC09	56329		Clock tempo-del-giorno: Secondi
DC0A	56330		Clock tempo-del-giorno: Minuti
DC0B	56331		Clock tempo-del-giorno: Ore+Indicatore AM/PM (bit 7)
DC0C	56332		Buffer dati I/O seriale sincrono
DC0D	56333		Registri controllo interruzione CIA (IRQ lettura/Maschera scrittura)
		7	Indicatore IRQ (1=Si e' verificata una IRQ)/Indicatore Imposta-Azzera
		4	IRQ indicatore 1 (lettura cassetta/ Input SRQ del bus seriale)
		3	Interruzione porta seriale
		2	Interruzione allarme clock tempo-del-giorno
		1	Interruzione Timer B
		0	Interruzione Timer A
DC0E	56334		Registro A di controllo del CIA
		7	Frequenza del clock tempo-del-giorno 1=50 Hz, 0=60 Hz
		6	Modo I/O porta seriale 1=Output, 0=Input
		5	Conteggio Timer A: 1=Segnali di CNT, 0=Clock O2 di sistema
		4	Caricamento forzato Timer A - 1=Si
		3	Modo funzionamento del Timer A 1=Monostabile, 0=Continuo
		2	Modo uscita per PB6 del Timer A 1=Bistabile, 0=Pulsazione
		1	Uscita per PB6 del Timer A-1=Si,0=No
		0	Attiva/Ferma Timer A-1=Attiva,0=Ferma
DC0F	56335		Registro B di controllo del CIA
		7	Imposta Allarme/Clock TOD 1=Allarme, 0=Clock
		6-5	Seleziona modo Timer B: 00 = Conteggio pulsazioni clock O2 di sistema 01 = Conteggio transizioni positive di CNT 10 = Conteggio pulsazioni underflow Timer A 11 = Conteggio pulsazioni underflow Timer A mentre CNT e' positivo
		4-0	Come registro A di controllo del CIA, ma per Timer B
DD00-DDFF	56576-56831		Adattatore interfaccia Complessa (CIA) #2 del 6526 MOS
DD00	56576		Porta dati A (bus seriale, RS-232, Controllo Memoria del VIC)
		7	Ingresso dati bus seriale
		6	Ingresso pulsazioni clock bus seriale
		5	Uscita dati bus seriale
		4	Uscita pulsazioni clock bus seriale
		3	Uscita segnale ATN del bus seriale
		2	Uscita dati dell'RS-232

			(porta Utente)
		1-0	Selezione banco di memoria di sistema del circuito VIC (default=11)
DD01	56577		Porta dati B (porta Utente, RS-232)
		7	Insieme dati Utente/RS-232 disponibile
		6	Azzera per invio Utente/RS-232
		5	Utente
		4	Scoperta riporto Utente/RS-232
		3	Indicatore anello Utente/RS-232
		2	Terminale dati Utente/RS-232 disponibile
		1	Richiesta invio Utente/RS-232
		0	Dati ricevuti Utente/RS-232
DD02	56578		Registro direzione dati - Porta A
DD03	56579		Registro direzione dati - Porta B
DD04	56580		Timer A: byte basso
DD05	56581		Timer A: byte alto
DD06	56582		Timer B: byte basso
DD07	56583		Timer B: byte alto
DD08	56584		Clock tempo-del-giorno: decimi di secondo
DD09	56585		Clock tempo-del-giorno: secondi
DD0A	56586		Clock tempo-del-giorno: minuti
DD0B	56587		Clock tempo-del-giorno: ore+AM/PM (bit 7)
DD0C	56588		Buffer dati I/O seriale sincrono
DD0D	56589		Registro controllo interruzione CIA (Legge NMI/Scrive la Maschera)
		7	Indicatore NMI (1=Si e' verificato un NMI)/Indicatore imposta-azzera
		4	NMI indicatore 1 (Ingresso dati ricevuti Utente/RS-232)
		3	Interruzione porta seriale
		1	Interruzione Timer B
		0	Interruzione Timer A
DD0E	56590		Registro A di controllo del CIA
		7	Frequenza del clock tempo-del-giorno 1=50 Hz, 0=60 Hz
		6	Modo di I/O della porta seriale 1=Input, 0=Output
		5	Conteggio Timer A - 1=Segnali di CNT, 0=Clock 02 di sistema
		4	Caricamento forzato Timer A - 1=Si
		3	Modo funzionamento Timer A 1=Monostabile, 0=Continuo
		2	Modo uscita su PB6 del Timer A 1=Bistabile, 0=Pulsazione
		1	Uscita su PB6 del Timer A 1=Si, 0=No
		0	Attiva/Ferma Timer A-1=Attiva,0=Ferma
DD0F	56591		Registro B di controllo del CIA
		7	Imposta Allarme/Clock TOD 1=Allarme, 0=Clock
		6-5	Seleziona il modo del Timer B: 00 = Conteggio pulsazioni clock 02 di sistema

			01 = Conteggio transizioni positive di CNT 10 = Conteggio pulsazioni underflow del Timer A 11 = Conteggio pulsazioni underflow Timer A mentre CNT e' positivo
		4-0	Come registro di controllo A del CIA, ma per Timer B
DE00-DEFF	56832-57087		Riservati a future espansioni di I/O
DF00-DFFF	57088-57343		Riservati a future espansioni di I/O

CAPITOLO 6

guida all'input/output

- Introduzione
- Output su TV
- Output su altri dispositivi
- Porte Giochi
- Descrizione dell'Interfaccia RS-232
- Porta Utente
- Bus Seriale
- Porta Espansione
- Cartuccia con Microprocessore Z-80

INTRODUZIONE

Le capacita' fondamentali degli elaboratori sono tre: calcolare, prendere decisioni e comunicare. Il calcolo e' probabilmente la cosa piu' facile da programmare, essendo familiari la maggior parte delle regole della matematica. Prendere delle decisioni non e' cosa troppo difficile, poiche' le regole della logica sono relativamente poche.
L'aspetto piu' complesso e' la comunicazione, perche' coinvolge il piu' piccolo insieme di leggi ben definite. Questa non e' una tascuratezza del progetto del computer: le regole offrono un'enorme flessibilita' nel comunicare virtualmente qualunque cosa, nei diversi modi possibili. L'unica vera regola e' la seguente: qualsiasi fonte di informazione deve presentare l'informazione stessa in maniera comprensibile al ricevitore.

OUTPUT SU TV

La forma piu' semplice di output messa a disposizione dal linguaggio BASIC e' l'istruzione PRINT: essa utilizza come dispositivo di output lo schermo della TV; gli occhi invece sono dispositivi di input, con i quali si sfrutta l'informazione presente sullo schermo.
Obiettivo principale della scrittura (con l'istruzione **PRINT**) **sullo** schermo e' la costruzione dell'informazione in modo che **questa risulti** facile da leggere. Si deve cercare di pensare come **gli artisti** grafici, usando i colori, posizionando lettere **maiuscole e minuscole**, ricorrendo pure alla grafica per comunicare **l'informazione nel** migliore dei modi. Basta ricordare che non e' **importante l'eleganza** del programma, quanto piuttosto la sua **capacita' di far capire il** significato dei risultati.
L'istruzione PRINT usa numerosi codici **carattere come "comandi" per** il cursore. Il tasto CRSR **non visualizza nulla: permette solamente al** cursore di cambiare posizione. **Altri comandi cambiano i colori**, puliscono lo schermo, **ed inseriscono o tolgono gli spazi. Il tasto** RETURN ha codice carattere **(CHR$) 13; la tabella completa di questi** codici e' **contenuta nell'Appendice C.**
Nel linguaggio BASIC **ci sono due funzioni che operano assieme alla** funzione PRINT: TAB, **che posiziona il cursore sulla posizione** assegnata a partire **dal margine sinistro dello schermo, e SPC, che** sposta il cursore **verso destra di un dato numero di spazi a partire** dalla posizione **attuale.**
I due punti (:) **nell'istruzione PRINT servono a separare ed a costruire l'informazione, il punto e virgola (;) separa due voci senza** alcuno spazio **tra di loro. Se quest'ultimo e' l'ultimo carattere della** linea, **il cursore rimane sulla linea appena stampata senza andare a capo alla linea successiva, sopprimendo (o sostituendo) cosi' il carattere di RETURN, normalmente stampato a fine linea.**
La virgola **separa dati di stampa all'interno delle colonne. Il COMMODORE 64 ha sullo schermo 4 colonne di 10 caratteri ciascuna; quando viene incontrata una virgola, il computer sposta il cursore all'inizio della colonna successiva. Come per il punto e virgola, se questo e' l'ultimo carattere della linea, il RETURN viene soppresso.**

Gli apici distinguono il testo letterale delle variabili; il primo apice delinea l'inizio dell'area letterale, il secondo la fine. Per inciso, non si deve riportare un apice conclusivo alla fine di una linea.
Il codice di RETURN (codice CHR$ 13) fa si' che il cursore si posizioni sulla prossima linea logica dello schermo, che non e' sempre la linea immediatamente successiva. Quando si digita oltre la fine della linea, questa viene concatenata alla linea successiva, per cui il computer sa che entrambi le linee costituiscono in realta' una sola linea di programma. Questi "agganci" sono contenuti nella tabella di "aggancio" delle linee (la costruzione di questa tabella e' spiegata nella mappa della memoria).
Una linea logica puo' essere costituita da una o due linee dello schermo, a seconda di cio' che e' stato digitato o stampato. La linea logica su cui si trova il cursore determina il punto dove il tasto RETURN invia il cursore stesso. La linea logica all'inizio dello schermo determina se il video avanza di una o due linee alla volta.
Ci sono altri modi di usare la TV come dispositivo di output. Il capitolo sulla grafica descrive i comandi che servono per creare oggetti in movimento sullo schermo. La sezione sul circuito VIC illustra come si possono cambiare dimensioni, colori e contorno dello schermo. Il capitolo sul suono mostra come l'altoparlante della TV possa creare musica ed effetti speciali.

OUTPUT SU ALTRI DISPOSITIVI

Spesso e' necessario inviare output su dispositivi diversi dallo schermo, quali registratori, stampanti, unita' a disco o modem. L'istruzione OPEN del BASIC crea un "canale" di colloquio con uno di uesti dispositivi. Una volta che tale canale e' stato aperto, l'istruzione PRINT# invia caratteri a quel dispositivo.

ESEMPIO DI ISTRUZIONI OPEN E PRINT :

```
100 OPEN 4,4:PRINT#4, "SCRITTURA SU STAMPANTE"
110 OPEN 3,8,3,"0:DISK-FILE,S,W":PRINT#3, "INVIATO A DISCO"
120 OPEN 1,1,1,"TAPE-FILE":PRINT#1, "SCRITTURA SU NASTRO"
130 OPEN 2,2,0,CHR$(10):PRINT#2, "INVIATO A MODEM"
```

L'istruzione OPEN e' diversa per ciascun dispositivo. I parametri per l'istruzione OPEN relativi a ciascun dispositivo sono elencati nella seguente tabella:

TABELLA DEI PARAMETRI DELL'ISTRUZIONE OPEN

Formato: OPEN file#,dispositivo#, numero, stringa

DISPOSITIVO	NUMERO DEL DISPOSITIVO	NUMERO	STRINGA
REGISTRATORE	1	0=Input 1=Output 2=Output senza EOT	Nome del file
MODEM	2	0	Registri di controllo
SCHERMO	3	0,1	
STAMPANTE	4 o 5	0=Maiuscole/Grafica 7=Maiuscole/ Minuscole	Stampa il testo
DISCO	da 8 a 11	2-14=Canale dati 15=Comando Canale	Numero Drive, nome del file, tipo del file, lettura / scrittura Comando

OUTPUT SU STAMPANTE

La stampante e' un dispositivo simile allo schermo. Quando si inviano dati alla stampante, l'interesse principale e' quello di creare un formato di facile lettura. Questo compito e' agevolato dai caratteri "reverse", in doppia ampiezza, maiuscoli e minuscoli, come pure dalla grafica programmabile per punti.

La funzione SPC funziona per la stampante allo stesso modo dello schermo. Altrettanto non si puo' dire invece dell'istruzione TAB: essa infatti calcola la posizione attuale sulla linea basandosi sulla posizione del cursore sullo schermo, e non sulla carta.

L'istruzione OPEN usata per la stampante crea il canale di comunicazione, specificando anche quale insieme di caratteri viene usato, se quello "maiuscole e grafica" oppure quello "maiscole e minuscole".

ESEMPI DI ISTRUZIONE OPEN PER STAMPANTE:

```
OPEN 1,4    : REM MAIUSCOLE/GRAFICA

OPEN 1,4,7  : REM MAIUSCOLE/MINUSCOLE
```

Quando si lavora con un insieme di caratteri, si possono stampare singole linee usando l'apposito insieme di caratteri. Quando si lavora con l'insieme maiuscole/grafica, il carattere "cursore verso il basso" [CHR$(17)] predispone i caratteri all'insieme maiuscole/minuscole. Quando invece si lavora con l'insieme maiuscole/minuscole, il carattere "cursore verso l'alto" [CHR$(145)] permette di usare l'insieme maiuscole/grafica.

Altre funzioni speciali della stampante vengono controllate attraverso particolari codici carattere. Tutti questi codici sono semplicemente stampati come ogni altro carattere.

TABELLA DEI CODICI CARATTERE PER IL CONTROLLO DELLA STAMPANTE:

CODICE CHR$	SCOPO
10	Alimentazione linea
13	RETURN (alimentazione linea automatica su stampanti CBM)
14	Inizio carattere a doppia ampiezza
15	Termine carattere a doppia ampiezza
18	Inizio caratteri "reverse"
146	Termine caratteri "reverse"
17	Imposta l'insieme maiuscole/minuscole
145	Imposta l'insieme maiuscole/grafica
16	Tabula la posizione dei successivi due caratteri
27	Spostamento alla posizione del punto specificato
8	Inizio grafica programmabile per punti
26	Ripete i dati della grafica

Per dettagli relativi all'uso dei codici di comando si veda il manuale Commodore della stampante.

OUTPUT SU MODEM

Il modem e' un semplice dispositivo in grado di tradurre i codici carattere in impulsi acustici, e viceversa, permettendo cosi' al computer di comunicare per mezzo delle linee telefoniche. L'istruzione OPEN relativa al modem imposta i parametri per controllare la velocita' ed il formato dell'altro computer con il quale si vuole comunicare. La stringa inviata al termine dell'istruzione OPEN puo' contenere due caratteri.

Le posizioni dei bit del primo codice carattere determinano la trasmittanza (velocita' di manipolazione di una linea), il numero di bit del dato ed il numero di bit di stop. Il secondo codice e' opzionale, ed i suoi bit specificano la parita' ed il duplex della trasmissione. Per particolari specifici su questo dispositivo si veda la sezione sull'RS-232 oppure il manuale **VICMODEM** .

ESEMPIO DI ISTRUZIONE OPEN PER IL MODEM:

```
OPEN 1,2,0,CHR$(6)             :REM 300 BAUD
OPEN 2,2,0,CHR$(163 CHR$(112):REM 110 BAUD, ECC.
```

La maggior parte dei computer usa il Codice Standard Americano per l'Interscambio delle Informazioni (ASCII - American Standard Code for Information Interchange). Questo insieme standard di codici carattere e' un po' diverso dai codici usati dal COMMODORE 64. Quando il COMMODORE 64 deve comunicare con altri computer, i soui codici carattere devono essere tradotti nei corrispondenti codici ASCII. La tabella dei codici ASCII standard e' riportata in Appendice C.

L'output su modem e' un compito assolutamente complicato, eccezion fatta per la necessita' di traduzione dei caratteri. In ogni caso si deve conoscere bene il dispositivo ricevente, specialmente quando si scrivono programmi dove il computer "colloquia" con un altro computer senza l'intervento umano. Un esempio di quanto detto potrebbe essere un programma da terminale che digita automaticamente un numero o una parola d'ordine segreta. Per riuscire a fare cio', si devono contare attentamente i caratteri ed i RETURN altrimenti il computer,

ricevendo i caratteri, non saprebbe di che cosa farsene.

USO DEI REGISTRATORI A CASSETTA

I registratori hanno una capacita' di memorizzazione dati quasi illimitata: piu' infatti il nastro e' lungo e piu' informazioni puo' memorizzare. Tuttavia, la loro maggiore limitazione e' costituita dal tempo: piu' numerosi sono i dati memorizzati e piu' lungo e' il tempo necessario per la ricerca dei dati stessi.

Quando si lavora con memorizzazioni su nastro e' necessario provare a minimizzare il fattore tempo. La pratica piu' comune consiste nel leggere l'intero file dati da cassetta nella memoria RAM, quindi elaborare e riscrivere tutti i dati su nastro. Questo procedimento permette di eseguire sort, editazioni e controlli sui dati, limitando pero' la dimensione dei file in base alla RAM disponibile.

Se il file dati e' piu' grande della memoria RAM disponibile, e' decisamente preferibile optare per i floppy disk, che permettono la lettura di dati a partire da qualunque posizione, senza la necessita' di leggere tutti i dati precedenti a quello cercato. Inoltre, si possono scrivere dati sopra altri dati piu' vecchi senza perturbare l'ordine della rimanente parte del file. Per questo il disco viene usato in tutte le applicazioni commerciali, quali i libri mastri ed i registri della corrispondenza.

L'istruzione PRINT# formatta i dati allo stesso modo dell'istruzione PRINT: anche tutta la punteggiatura si comporta allo stesso modo. Bisogna pero' tenere ben presente che non si sta lavorando con lo schermo. La formattazione deve essere eseguita tenendo ben presente l'istruzione INPUT#.

Consideriamo l'istruzione INPUT#1,A$,B$,C$. Quando viene usata con lo schermo, le virgole di separazione delle variabili inseriscono spazi bianchi sufficienti a sistemare ogni elemento su una colonna ampia 10 caratteri. Su cassetta, invece, vengono aggiunti da 1 a 10 spazi, a seconda della lunghezza delle stringhe, comportando cosi' uno spreco di spazio sul nastro.

Ancora peggiore e' cio' che accade quando l'istruzione INPUT# cerca di leggere queste stringhe: l'istruzione INPUT#1,A$,B$,C$ non trova alcun dato per B$ e C$, mentre A$ contiene tutte e tre le variabili separate dagli spazi bianchi. Vediamo che cosa e' accaduto al file su nastro:

```
A$="DOG" B$="CAT" C$="TREE"
PRINT# 1, A$, B$, C$
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

D O G                C  A  T                    T  R  E  E  RETURN
```

L'istruzione INPUT# funziona come l'istruzione INPUT: quando si digitano i dati nell'istruzione INPUT, i dati vengono separati dal tasto RETURN oppure dalle virgole. L'istruzione PRINT# inserisce, alla fine della linea, un RETURN, proprio come l'istruzione PRINT. A$ contiene tutti e tre i valori perche' sul nastro non e' presente alcun separatore fra essi (il separatore compare solo alla fine di tutti e tre i valori).

I separatori piu' indicati per il nastro sono la virgola e RETURN ; il codice di quest'ultimo viene inserito automaticamente alla fine dell'istruzione PRINT o PRINT#. Un modo per inserire il codice di RETURN fra gli elementi e' quello di usare solamente una voce per

l'istruzione PRINT#. Un modo ancora migliore consiste nell'impostare una variabile al codice CHR$ di RETURN [CHR$(13)], oppure nell'uso di una virgola. In quest'ultimo caso l'istruzione e' R$=",":PRINT#1,A$ R$ B$ R$ C$. Non si devono inserire virgole o qualsiasi altro carattere di punteggiatura tra i nomi delle variabili, perche' il COMMODORE 64 considera tali variabili separatamente, provocando cosi' uno spreco di spazio nel programma.

Un file registrato su nastro in maniera corretta deve essere simile al seguente:

```
1 2 3 4 5 6 7 8 9 10 11 12 13

D O G , C A T , T R  E  E  RETURN
```

L'istruzione GET# preleva dati da nastro un carattere alla volta, compreso il codice di RETURN e di tutto il resto della punteggiatura. Il codice CHR$(0) viene interpretato come stringa vuota, non come una stringa di un carattere di codice 0. Il tentativo di usare la funzione ASC con una stringa vuota si risolve nel messaggio di errore **ILLEGAL QUANTITY ERROR.**

La riga GET#1,A$:A=ASC(A$) viene usata comunemente per esaminare da programma i dati registrati sul nastro. Per evitare messaggi di errore, la precedente riga puo' essere modificata nel modo seguente:

```
GET#1,A$ : A=ASC(A$+CHR$(0))
```

CHR$(0) posto al termine della stringa mette al riparo da eventuali stringhe vuote, ma non interessa la funzione ASC quando A$ contiene altri caratteri.

MEMORIZZAZIONI DI DATI SU FLOPPY DISK

I dischetti permettono tre diverse forme di memorizzazione. I file sequenziali si comportano come quelli su nastro, ma se ne possono adoperare contemporaneamente piu' di uno. I file relativi consen ono di organizzare i dati in record, e quindi di leggere ed allocare individualmente i record nel file. I file random consentono di lavorare con dati memorizzati in qualunque zona del disco; sono organizzati in ssegmenti di 256 byte chiamati blocchi.

Le limitazioni dell'istruzione PRINT# su disco sono analoghe a quelle riguardanti i nastri. Anche in questo caso sono necessarie le virgole o i RETURN per separare i dati, e CHR$(0) viene ancora letto dall'istruzione GET# come stringa vuota.

I file relativi e quelli random usano entrambi dati separati e comandi di "canale". I dati scritti su disco attraversano il canale dati, dove vengono memorizzati in un buffer transiente situato nella RAM del disco. Quando il blocco e' completo, attraverso un canale di comando viene inviato un comando che comunica al drive dove inserire i dati, quindi l'intero buffer viene scritto.

Applicazioni richiedenti un grande numero di dati da elaborare trovano adeguata sistemazione nei file relativi su disco. Una simile organizzazione richiede poco tempo di elaborazione pur garantendo una buona flessibilita' al programma. Una guida completa per la programmazione e l'uso dei file su disco e' riportata nel manuale dell'unita' disco.

PORTE-GIOCHI

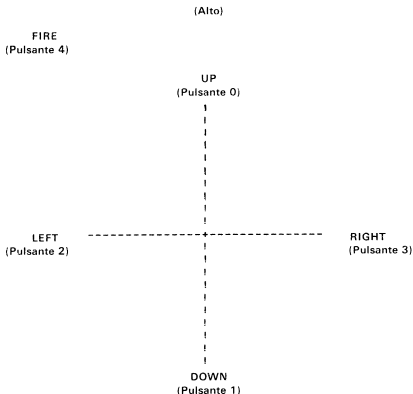
Il COMMODORE 64 ha due Porte Giochi a 9 pin che permettono l'uso di joystick, paddle e penna ottica. Ogni porta accetta un joystick o una paddle. La porta A e' l'unica a consentire l'inserimento della penna ottica da adibire a particolari controlli grafici, ecc. Questo paragrafo riporta vari esempi dell'uso di joystick e paddle sia da BASIC che da linguaggio macchina.

Il joystick digitale e' collegato con il CIA#1 (Adattatore Interfaccia Complessa 6526 MOS). Questo dispositivo di input/output gestisce anche i pulsanti di sparo della paddle ed esegue la scansione della tastiera. Il circuito CIA 6526 ha 16 registri locati in memoria da 56320 a 56335 ($DC00-$DC0E HEX). I dati della Porta A compaiono nella locazione 56320 ($DC00 HEX), quelli della Porta B nella locazione 56321 ($DC01 HEX).

Un joystick digitale ha cinque pulsanti distinti, quattro dei quali sono usati per le direzioni ed uno come pulsante di sparo. La disposizione dei pulsanti del joystick e' la seguente:

```
                              (Alto)

        FIRE
     (Pulsante 4)

                                UP
                           (Pulsante 0)
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
       LEFT      ----------------+----------------      RIGHT
    (Pulsante 2)                 !                   (Pulsante 3)
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                                 !
                               DOWN
                           (Pulsante 1)
```

Questi interruttori corrispondono ai 5 bit bassi del dato contenuti nelle locazioni 56320 o 56321. Normalmente, un bit e' impostato a 1 se NON viene scelta la direzione o se NON viene premuto il pulsante di sparo. Quando invece quest'ultimo pulsante viene premuto, il corrispondente bit (bit 4, in questo caso) cambia e passa a 0. La lettura da BASIC del joystick puo' essere ottenuta usando la sottoprocedura seguente:

```
10 FORK=0TO10:REM IMPOSTA STRINGA DIREZIONE
20 READDR$(K):NEXT
30 DATA"","N","S","","W","NW"
40 DATA"SW","","E","NE","SE"
50 PRINT"GOING...";
60 GOSUB100:REM LEGGE IL JOYSTICK
65 IFDR$(JV)=""THEN80:REM CONTROLLA SE SI E' SCELTA UNA DIREZIONE
70 PRINTDR$(JV);" ";:REM STAMPA QUALE DIREZIONE
80 IFFR=16THEN60:REM CONTROLLA SE E' STATO PREMUTO IL PULSANTE DI
81 REM                SPARO
90 PRINT"-----F-----I-----R-----E-----!!!":GOTO60
100 JV=PEEK(56320):REM PRELEVA IL VALORE DEL JOYSTICK
110 FR=JVAND16:REM PREPARA LO STATO DEL PULSANTE DI SPARO
120 JV=15-(JVAND15):REM PREPARA IL VALORE DELLA DIREZIONE
130 RETURN
```

NOTA: Per il secondo joystick impostare JV=PEEK(56321)

I valori di JV corispondono alle seguenti direzioni:

JV	DIREZIONE
0	Nessuna
1	Alto
2	Basso
3	-
4	Sinistra
5	Alto e sinistra
6	Basso e sinistra
7	-
8	Destra
9	Alto e destra
10	Basso e destra

Lo stesso compito viene svolto dalla seguente sottoprocedura:

```
1000 .PAGE (JOYSTICK.8/5)  JOYSTICK - LETTURA PULSANTE DI SPARO
1010 ;
1020 ;AUTORE - BILL HINDORFF
1030 ;
1040 DX=$C110
1050 DY=$C111
1060 *=$C200
1070 DJRR  LDA $DC00    ; PRELEVA L'INPUT DALLA SOLA PORTA A
1080 DJRRB LDY #0       ; QUESTA ROUTINE LEGGE E DECODIFICA I DATI
1090       LDX #0       ; DEL PULSANTE DI SPARO DEL JOYSTICK IN INPUT
1100       LSR A        ; ALL'ACCUMULATORE. I 5 BIT MENO SIGNIFICATIVI
1110       BCS DJR0     ; CONTENGONO L'INFORMAZIONE DI CHIUSURA
1120       DEY          ; DEL PULSANTE. SE UN PULSANTE VIENE CHIUSO,
1130 DJR0  LSR A        ; SI PRODUCE UN BIT 0, SE VIENE APERTO SI
1140       BCS DJR1     ; PRODUCE UN BIT 1. LE DIREZIONI DEL JOYSTICK
1150       INY          ; SONO DESTRA, SINISTRA, AVANTI, INDIETRO
1160 DJR1  LSR A        ; BIT3=DESTRA, BIT2=SINISTRA, BIT1=AVANTI,
1170       BCS DJR2     ; BIT0=INDIETRO, BIT4=FUOCO.
1180       DEX          ; AL TEMPO RTS, DX E DY CONTENGONO IL
                        ; COMPLEMENTO
1190 DJR2  LSR A        ; A 2 DEI NUMERI DI DIREZIONE, CIOE' $FF=-1,
```

```
1200        BCS DJR3    ; $00=0,$01=1. DX=1 (DESTRA),DX=-1 (SINISTRA),
1210        INX         ; DX=0 (NESSUN CAMBIAMENTO DI X), DY=-1 (ALTO)
1220 DJR3   LSR A       ; D1=1 (BASSO), DY=0 (NESSUN CAMBIAMENTO DI Y)
1230        STX DX      ; LA POSIZIONE "INDIETRO" DEL JOYSTICK
                        ; CORRISPONDE
1240        STY DY      ; AL MOVIMENTO VERSO L'ALTO DELLO SCHERMO, LA
1250        RTS         ; POSIZIONE "AVANTI" AL MOVIMENTO VERSO IL
1260 ;                    BASSO DELLO SCHERMO. AL TEMPO RTS,
1270 ; L'INDICATORE DI RIPORTO CONTIENE LO STATO DEL PULSANTE
1280 ; DI SPARO. SE C=1 IL PULSANTE NON E' PREMUTO, SE C=0 E' PREMUTO
1290 ;
1300 .END
```

PADDLES

Una paddle puo' essere connessa sia al circuito CIA#1 che al circuito SID (Dispositivo Interfaccia del Suono 6581 MOS) attraverso una porta giochi. Il valore della paddle e' letto attraverso i registri SID 54297 ($D419 HEX) e 54298 ($D41A HEX). LA LETTURA ESCLUSIVAMENTE DA BASIC DELLE PADDLES NON E' ATTENDIBILE!!!! Il modo migliore di usare le paddle, da BASIC o da codice macchina, e' usare la seguente sottoprocedura in linguaggio macchina (accesso da BASIC consentito dall'istruzione SYS, quindi lettura tramite l'istruzione PEEK delle locazioni di memoria usate dalla sottoprocedura)...

```
1000 ;*********************************************************
1010 ;* ROUTINE DI LETTURA DI 4 PADDLE (UTILIZZABILE ANCHE PER 2) *
1020 ;*********************************************************
1030 ; AUTORE - BILL HINDORFF
1040 PORTA=$DC00
1050 CIDDRA=$DC02
1060 SID=$D400
1070 *=$C100
1080 BUFFER *=*+1
1090 PDLX *=*+2
1100 PDLY *=*+2
1110 BTNA *=*+1
1120 BTNB *=*+1
1130 *=$C000
1140 PDLRD
1150     LDX #1          ; PER 4 PADDLE OPPURE 2 JOYSTICK
1160 PDLRD0              ; PUNTO DI INGRESSO PER UNA COPPIA
1170     SEI
1180     LDA CIDDRA      ; PRELEVA IL VALORE CORRENTE DI DDR
1190     STA BUFFER      ; E LO SALVA
1200     LDA #$C0
1210     STA CIDDRA      ; IMPOSTA LA PORTA A PER L'INPUT
1220     LDA #$80
1230 PDLRD1
1240     STA PORTA       ; INDIRIZZA UNA COPPIA DI PADDLE
1250     LDY #$80        ; BREVE PAUSA
1260 PDLRD2
1270     NOP
1280     DEY
1290     BPL PDLRD2
1300     LDA SID+25      ; PRELEVA IL VALORE DI X
1310     STA PDLX,X
```

```
1320     LDA SID+26      ; PRELEVA IL VALORE DI Y
1330     STA PDLY,X
1340     LDA PORTA       ; TEMPO LETTURA PULSANTI DI SPARO DELLA PADDLE
1350     ORA #$80        ; COME SOPRA, PER LA SECONDA COPPIA
1360     STA BTNA        ; BIT 2 CORRISPONDE A PDL X E BIT 3 A PDL Y
1370     LDA #$40
1380     DEX             ; SONO STATE LETTE TUTTE LE COPPIE ?
1390     BPL PDLRD1      ; NO
1400     LDA BUFFER
1410     STA CIDDRA      ; RIPRISTINA IL PRECEDENTE VALORE DI DDR
1420     LDA PORTA+1     ; PER LA SECONDA COPPIA -
1430     STA BTNB        ; BIT 2 CORRISPONDE A PDL X E BIT 3 A PDL Y
1440     CLI
1450     RTS
1460 .END
```

Le paddle possono essere lette usando il seguente programma BASIC:

```
10 C=12*4096:REM IMPOSTA IL PUNTO DI PARTENZA DELLA SOTTOPROCEDURA
11 REM SCRIVE TRAMITE POKE NELLA SOTTOPROCEDURA DI LETTURA
12 REM DELLE PADDLE
15 SYSC:REM RICHIAMA LA SOTTOPROCEDURA DELLE PADDLE
30 P1=PEEK(C+257):REM IMPOSTA IL VALORE DELLA PADDLE 1
40 P2=PEEK(C+258):REM  "    "     "  "    "      "   2
50 P3=PEEK(C+259):REM  "    "     "  "    "      "   3
60 P4=PEEK(C+260):REM  "    "     "  "    "      "   4
61 REM LEGGE LO STATO DEL PULSANTE DI SPARO
62 S1=PEEK(C+261):S2=PEEK(C+262)
70 PRINTP1,P2,P3,P4:REM STAMPA I VALORI DELLE PADDLE
72 REM STAMPA LO STATO DEL PULSANTE DI SPARO
75 PRINT:PRINT"FIRE A ";S1;"FIRE B ";S2
80 FORW=1TO50:NEXT:REM BREVE ATTESA
         SHIFT CLR HOME
90 PRINT"♥":PRINT:GOTO20:REM AZZERA LO SCHERMO E RICOMINCIA
95 REM DATI PER SOTTOPROCEDURA IN LINGUAGGIO MACCHINA
100 DATA162,1,120,173,2,220,141,0,193,169,192,141,2,220,169
110 DATA128,141,0,220,160,128,234,136,16,252,173,25,212,157
120 DATA1,193,173,26,212,157,3,193,173,0,220,9,128,141,5,193
130 DATA169,64,202,16,222,173,0,193,141,2,220,173,1,220,141
140 DATA6,193,88,96
```

PENNA OTTICA

L'ingresso penna ottica registra in un circuito "latch", sul fianco di un impulso in caduta, la corrente posizione dello schermo, utilizzando una coppia di registri (LPX, LPY). Il registro 19 ($13 HEX) posizione X contiene gli 8 MSB della posizione X all'istante della transizione. Poiche' la posizione X e' definita da un contatore a 512 posizioni (9 bit), viene fornita una risoluzione di due punti orizzontali. Analogamente, la posizione Y viene registrata nel circuito "latch" del registro 20 ($14 HEX); in questo caso, gli 8 bit forniscono, all'interno dello schermo visibile, una risoluzione di quadro singola. Il circuito "latch" della penna ottica puo' essere triggerato solamente una volta per quadro, per cui tutti gli scatti seguenti non hanno alcun effetto. Occorre percio' eseguire diverse prove (mediamente da tre in su) prima di iniziare ad operare sullo schermo con la penna ottica; il numero di prove da eseguire varia

tuttavia in base alle caratteristiche della penna ottica impiegata.

DESCRIZIONE DELL'INTERFACCIA RS-232

SCHEMA GENERALE

Il COMMODORE 64 incorpora un'interfaccia per il collegamento con un qualunque modem, stampante o altri dispositivi compatibili con l'RS-232. Per collegare un dispositivo al COMMODORE 64 occorrono una certa capacita' ed un po' di programmazione.

L'RS-232 installata sul COMMODORE 64 risponde al formato RS-232 standard, ma le tensioni elettriche sono di livello TTL (0...5V) piuttosto che dell'intervallo -12...+12 V. Le necessarie conversioni della tensione di voltaggio tra il COMMODORE 64 e l'RS-232 sono svolte dalla cartuccia interfaccia Commodore RS-232.

L'accesso al software dell'interfaccia RS-232 puo' avvenire sia da BASIC che da KERNAL, per la programmazione in linguaggio macchina.

A livello BASIC, l'RS-232 usa i normali comandi del BASIC: OPEN, CLOSE, CMD, INPUT#, GET#, PRINT#; le variabili riservate ST.INPUT# e GET# vanno a prelevare i dati dal buffer ricevente, mentre PRINT# e CMD vanno a sistemare i dati nel buffer trasmittente. Piu' avanti in questo capitolo sara' spiegato piu' dettagliatamente l'uso di questi comandi.

I gestori dell'RS-232 a livello di bit e byte del KERNAL funzionano sotto il controllo dei timer e delle interruzioni del dispositivo 6526 CIA #2. Un'elaborazione RS-232 genera all'interno del circuito 6526 una serie di richieste NMI (Non Maskable Interrupt - Interruzioni Non Mascherabili). Questo permette un'elaborazione RS-232 di fondo durante l'esecuzione di programmi BASIC ed in linguaggio macchina. Per evitare la distruzione di dati memorizzati, o per prevenirne la trasmissione attraverso le NMI generate dalle sottoprocedure dell'RS-232, all'interno delle sottoprocedure del KERNAL, del registatore e del bus seriale sono state previste particolari aree di memoria riservate. Durante le attivita' del registratore o del bus seriale, NON si possono ricevere dati da dispositivi RS-232. Ma, poiche' tali aree di memoria riservata sono solamente locali (supponendo che la programmazione sia stata accorta), non si genera alcuna interferenza.

Nell'interfaccia RS-232 del COMMODORE 64 ci sono due buffer che aiutano a prevenire eventuali perdite di dati durante la ricezione o la trasmissione di informazioni RS-232.

I buffer RS-232 del KERNAL del COMMODORE 64 sono costituiti da due buffer "FIFO" (First In, First Out - primo entrato, primo uscito), ciascuno lungo 256 byte, situati all'inizio della memoria. L'apertura di un canale RS-232 alloca automaticamente per questi buffer 512 byte di memoria. Se oltre la fine di un programma BASIC non c'e' spazio libero a sufficienza, non viene stampato alcun messaggio di errore, ma la fine del programma viene distrutta. Quindi, ATTENZIONE!

Con il comando CLOSE questi buffer vengono automaticamente rimossi dalla memoria.

APERTURA DI UN CANALE RS-232

Non si puo' tenere aperto piu' di un canale RS-232 alla volta: una seconda istruzione OPEN causa infatti la nuova impostazione dei puntatori al buffer, provocando di conseguenza la perdita di tutti i

caratteri contenuti nel buffer trasmittente o ricevente.
Il campo nome del file puo' contenere fino a 4 caratteri: i primi due sono i caratteri di controllo e di comando dei registri, gli altri due sono riservati alle future opzioni di sistema. velocita' di trasmissione (espressa in baud), parita' ed altre opzioni possono essere selezionate attraverso questa caratteristica.
Sulla parola di controllo non viene eseguito alcun controllo di errore per scoprire una trasmittanza non implementata. Ciascuna parola di controllo illecita fa si' che l'output di sistema lavori a velocita' molto basse (al di sotto di 50 baud).

SINTASSI BASIC:

OPEN lfn,2,0, "<registro di controllo> <registro di comando> <opt baud low> <opt baud high>"

LFN – Numero di file logico; e' un qualunque numero compreso fra 1 e 255. Da notare che un numero di file maggiore di 127 comporta un avanzamento automatico delle linee dopo ogni ritorno carrello.

LUNGHEZZA DELLA PAROLA		
BIT 6	BIT 5	LUNGHEZZA PAROLA DATI
0	0	8 Bit
0	1	7 Bit
1	0	6 Bit
1	1	5 Bit

3	2	1	0	TRASMITTANZA
0	0	0	0	Utente
0	0	0	1	50 Baud
0	0	1	0	75
0	0	1	1	110
0	1	0	0	134.5
0	1	0	1	150
0	1	1	0	300
0	1	1	1	600
1	0	0	0	1200
1	0	0	1	(1800) 2400
1	0	1	0	2400
1	0	1	1	3600 [NI]
1	1	0	0	4800 [NI]
1	1	0	1	7200 [NI]
1	1	1	0	9600 [NI]
1	1	1	1	19200 [NI]

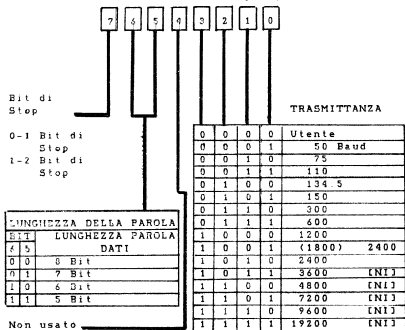



Figura 6.1 - Mappa del registro di controllo

<registro di controllo> – Carattere contenuto in un singolo byte (vd. fig. 6.1), richiesto per specificare la trasmittanza. Se il piu' basso dei 4 bit della trasmittanza e' uguale a zero, i caratteri <opt baud low> e <opt baud high> forniscono una tasmittanza calcolata come segue:

<opt baud low> = <frequenza di sistema/velocita'/2-100 - <opt baud high> * 256

(opt baud high) = INT[(frequenza di sistema/velocita'/2-100) / 256]

BIT 7	BIT 6	BIT 5	OPERAZIONI	
	-	-	0	Parita' disabilitata Nessuna generazione/ ricezione
	0	0	1	Parita' dispari ricevente/trasmittente
	0	1	1	Parita' pari ricevente/trasmittente
	1	0	1	Segno trasmesso Controllo parita' disabilitato
	1	1	1	Spazio trasmesso Controllo parita' disabilitato

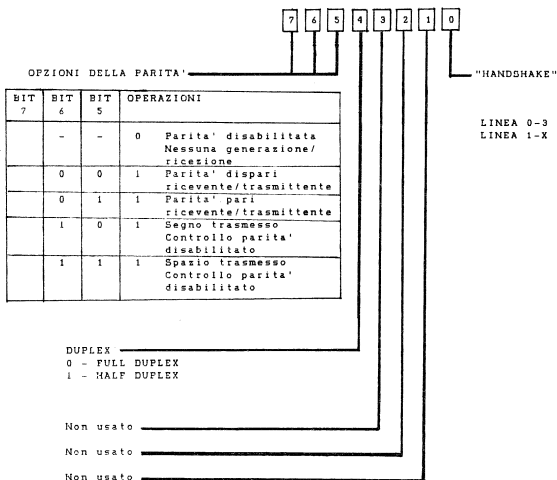



Figura 6.2 - Mappa del registro di comando

Le formule precedentemente riportate si basano sulle seguenti considerazioni:

Frequenza di sistema = 1.02273E6 NTSC (North American TV Standard -Standard TV nordamericano)
= 0.98525E6 PAL (Standard TV inglese e di molti paesi europei)

<registro di comando> - Carattere su singolo byte (vd. fig. 6.2) che definisce altri parametri del terminale. NON e' obbligatorio.

INGRESSO KERNAL

OPEN ($FFC0) (Per ulteriori informazioni sulle condizioni e sulle istruzioni di ingresso si veda la descrizione dettagliata del KERNAL).

NOTA IMPORTANTE: In un programma BASIC, il comando OPEN dell'RS-232 puo' essere effettuato prima di creare qualunque variabile o schiera, perche' viene eseguito un CLR automatico al momento dell'apertura di un canale RS-232 (cio' grazie ai 512 byte allocati all'inizio della memoria. Da ricordare che il programma viene distrutto se i 512 byte dello spazio non sono disponibili al momento dell'istruzione OPEN.

PRELIEVO DI DATI DA UN CANALE RS-232

Al momento del prelievo di dati da un canale RS-232, il buffer ricevente del COMMODORE 64 arresta l'ingresso dei caratteri al 255-esimo, prima che il buffer stesso vada in overflow. Questo fatto e' riportato nella parola di stato dell'RS-232 (ST in BASIC, RSSTAT in linguaggio macchina). Se si verifica un overflow, vengono persi tutti i caratteri ricevuti dopo che il buffer e' risultato pieno. E' quindi consigliabile tenere il buffer piu' vuoto possibile.
Se si desidera che la ricezione dei dati avvenga ad alta velocita' (il BASIC puo' solo andare velocemente, anche in considerazione del "garbage collection", e causare di conseguenza l'overflow del buffer ricevente), e' necessario usare sottoprocedure in linguaggio macchina per trattare questo tipo di flusso di dati.

SINTASSI BASIC:

Consigliata: GET# lfn, <variabile stringa>

SCONSIGLIATA: INPUT# lfn, <lista variabile>

INGRESSI KERNAL:

CHKIN ($FFC6) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

GETIN ($FFE4) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

CHRIN ($FFCF) - Per ulteriori informazioni sulle condizioni di ingresso ed uscita, si veda la mappa della memoria.

NOTE: * Se la lunghezza della parola e' inferiore a 8 bit, tutti i bit inutilizzati vengono impostati a zero.
* Se GET# non trova alcun dato nel buffer, ritorna un carattere nullo ("").
* Se si usa INPUT#, il sistema resta in condizione di attesa fino alla ricezione di un carattere nullo ed un successivo ritorno carrello. Tuttavia, se scompare la linea CTS (Clear To Send - Azzera per invio) o DSR (DataSsette Ready - registratore disponibile) durante INPUT#, il sistema rimane nello stato solo-ripristino. Ecco perche' NON sono consigliabili le sottoprocedure INPUT# e CHRIN.
La sottoprocedura CHKIN tratta l'"handshaking" sulla x-esima linea seguente lo standard EIA (agosto 1979) per l'interfaccia RS-232 (le linee RTS (Request To Send - richiesta di invio), CTS e DCD (segnale di linea ricevuta) sono implementate sul COMMODORE 64 definito come dispositivo terminale di dati).

INVIO DI DATI AD UN CANALE RS-232

Al momento dell'inoltro di dati occorre tener presente che il buffer di output puo' contenere 255 caratteri, dopodiche' si verifica un trabocco del buffer pieno (overflow). Il sistema attende, nella sottoprocedura CHROUT, che venga attivata la trasmissione, oppure che vengano premuti i tasti RUN/STOP e RESTORE per ripristinare il sistema con una "partenza a caldo".

SINTASSI BASIC:

CMD lfn - Agisce come riportato nelle specifiche del BASIC

PRINT#lfn, <lista variabile>

INGRESSI KERNAL:

CHKOUT ($FFC9) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

CHROUT ($FFD2) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

NOTE IMPORTANTI: All'interno del canale di output non avviene alcun ritardo del ritorno carrello: cio' significa che una normale stampante RS-232 non puo' stampare correttamente, a meno che non sia dotata di qualche area di memoria riservata (che richieda un'attesa al COMMODORE 64), o di un buffer interno. L'area di memoria riservata puo' essere facilmente implementata in un programma. Se si implementa un "handshake" CTS (linea X), il buffer del COMMODORE 64 viene riempito, permettendo quindi al dispositivo RS-232 di autorizzare l'area di memoria riservata all'invio di altro output finche' permane la condizione di trasmissione. L'"handshake" della linea X e' una procedura di "handshake" che usa linee multiple per la trasmissione e la ricezione di dati.

L'"handshake" della linea X viene trattato dalla sottoprocedura CHKOUT, che segue lo standard EIA (agosto 1979) per le intarfacce RS-232. Le linee RTS, CTS e DCD vengono implementate sul COMMODORE 64 definendo quest'ultimo come Dispositivo Terminale Dati.

CHIUSURA DI UN CANALE DATI RS-232

La chiusura di un file RS-232 provoca la perdita di tutti i dati (trasmessi, stampati oppure no) contenuti nei buffer al momento dell'esecuzione, l'arresto di tutte le attivita' di trasmissione e ricezione dell'RS-232, e la rimozione di tutti e due i buffer dell'RS-232.

SINTASSI BASIC:

CLOSE lfn

INGRESSO KERNAL:

CLOSE ($FFC3) - Per ulteriori informazioni sulle condizioni di ingresso e di uscita, si veda la mappa di memoria.

NOTA: Prima di chiudere un canale, assicurarsi che tutti i dati siano stati trasmessi; cio' si ottiene da BASIC usando la forma:

```
100 SS=ST: IF(SS=0 OR SS=8) THEN 100
110 CLOSE lfn
```

(Locazioni $DD00 - $DD0F del dispositivo 2 6526)

PIN	6526	DESCRIZIONE	EIA	ABBR.	IN/OUT	MODI
C	PB0	Dati ricevuti	(BB)	Sin	IN	1 2
D	PB1	Richiesta di invio	(CA)	RTS	OUT	1(*) 2
E	PB2	Terminale dati disponibile	(CD)	DTR	OUT	1(*) 2
F	PB3	Indicatore anello	(CE)	RI	IN	3
H	PB4	Ricevuto segnale di linea	(CF)	DCD	IN	2
J	PB5	Non asegnato	-	XXX	IN	3
K	PB6	Azzera per invio	(CB)	CTS	IN	2
L	PB7	Insieme dati disponibile	(CC)	DSR	IN	2
B	FLAG2	Dati ricevuti	(BB)	Sin	IN	1 2
M	PA2	Dati trasmessi	(BA)	Sout	OUT	1 2
A	GND	Linea di terra (protezione	(AA)	GND	-	1 2
N	GND	Terra del segnale	(AB)	GND	-	1 2 3

MODI:

1) Interfaccia linea 3 (Sin, Sout, GND)
2) Interfaccia linea X
3) Disponibile solo all'Utente (Non usata / non implementata nel codice)

*) Queste linee sono mantenute alte durante il modo LINEA-3.

Tabella 6.1 - Linee Porta Utente

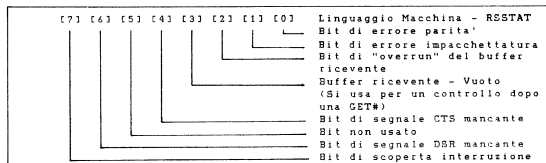



Figura 6.3 - Registro di stato dell'RS-232

NOTE:

* Se BIT=0, non si e' incontrato alcun errore.
* Il registro di stato dell'RS-232 puo' essere letto da BASIC usando la variabile ST.
* Se ST e' letta da BASIC oppure usando la sottoprocedura READST del KERNAL, la parola di stato dell'RS-232 viene azzerata all'uscita. Se e' la parola di stato e' necessaria piu' volte, ST deve essere assegnata ad un'altra variabile. Ad esempio:

```
SR=ST: REM ASSIGNS ST TO SR
```

* Lo stato dell'RS-232 viene letto (ed azzerato) solo quando il canale RS-232 e' stato l'ultimo I/O esterno usato.

ESEMPI DI PROGRAMMI BASIC

```
10 REM QUESTO PROGRAMMA INVIA/RICEVE DATI A/DA UN TERMINALE 700
11 REM SILENZIOSO MODIFICATO PER IL PET ASCII
20 REM PREDISPOSIZIONE DEL TERMINALE 700 SILENZIOSO: 300 BAUD, ASCII
21 REM A 7 BIT, PATITA' DEL CARATTERE, FULL DUPLEX
30 REM PREDISPOSIZIONE DEL COMPUTER: COME SOPRA, USANDO L'INTERFACCIA
31 DELLA LINEA 3
100 OPEN 2,2,3,CHR$(6+32)+CHR$(32+128):REM APERTURA DEL CANALE
110 GET#2,A$:REM ATTIVA IL CANALE RICEVENTE (LANCIA UN CARATTERE
111 REM          NULLO)
200 REM CICLO PRINCIPALE
210 GET B$:REM PRELIEVO DALLA TASTIERA DEL COMPUTER
220 IF B$ <> "" THEN PRINT#2,B$;:REM INVIA AL TERMINALE IL CARATTERE
225 REM                                   BATTUTO
230 GET#2,C$:REM PRELEVA UN CARATTERE DAL TERMINALE
240 PRINT B$;C$;:REM STAMPA TUTTI I CARATTERI IN INGRESSO ALLO
241 REM              SCHERMO DEL COMPUTER
250 SR=ST:IF SR=0 OR SR=8 THEN 200:REM CONTROLLA LO STATO, SE E'
260 REM                                       TUTTO A POSTO
260 REM MESSAGGIO DI ERRORE
310 PRINT "ERROR: ";
320 IF SR AND 1 THEN PRINT "PARITA'"
330 IF SR AND 2 THEN PRINT "PACCHETTO"
340 IF SR AND 4 THEN PRINT "BUFFER RICEVENTE PIENO"
350 IF SR AND 128 THEN PRINT "BREAK"
360 IF (PEEK(673) AND 1) THEN 360:REM ATTENDE TUTTI I CARATTERI
370 CLOSE 2:END
```

```
100 OPEN 5,2,3,CHR$(6)
110 DIM F%(255),T%(255)
200 FOR J=32 TO 64:T%(J)=J:NEXT
210 T%(13)=13:T%(20)=8:RV=18:CT=0
220 FOR J=65 TO 90:K=J+32:T%(J)=K:NEXT
230 FOR J=91 TO 95:T%(J)=J:NEXT
240 FOR J=193 TO 218:K=J-128:T%(J)=K:NEXT
250 T%(146)=16:T%(133)=16
260 FOR J=0 TO 255
270 K=T%(J)
280 IF K<>0 THEN F%(K)=J:F%(K+128)=J
290 NEXT
300 PRINT " "CHR$(147)
310 GET#5,A$
320 IF A$="" OR ST <> 0 THEN 360
330 PRINT " "CHR$(157);CHR$(F%(ASC(A$)));
340 IF F%(ASC(A$))=34  THEN POKE212,0
350 GOTO 310
360 PRINT CHR$(RV)" "CHR$(157);CHR$(146);:GET A$
370 IF A$ <> "" THEN PRINT#5,CHR$(T%(ASC(A$)));
380 CT=CT+1
390 IF CT=8 THEN CT=0:RV=164-RV
410 GOTO 310
```

PUNTATORI ALLA LOCAZIONE DI BASE DEL BUFFER RICEVENTE/TRASMITTENTE

$00F7-RIBUF Puntatore composto da due byte per la locazione di base del buffer ricevente.
$00F9-ROBUF Puntatore composto da due byte per la locazione di base del buffer trasmittente.

Queste due locazioni sono impostate dalla sottoprocedura OPEN del KERNAL; ciascuna punta ad un differente buffer di 256 byte. Queste locazioni vengono disallocate quando la sottoprocedura CLOSE del KERNAL imposta uno zero nel byte di ordine piu' alto ($00F8 e $00F9 HEX); la loro allocazione e disallocazione e' consentita anche a chi programma in linguaggio macchina per scopi propri, ed e' ottenibile rimuovendo o creando solamente i buffer richiesti. L'allocazione di questi buffer eseguita attraverso il linguaggio macchina richiede una certa attenzione nel corretto posizionamento dei puntatori all'inizio della memoria, specialmente se nello stesso tempo sono presenti in macchina altri programmi BASIC.

LOCAZIONI DELLA MEMORIA DI PAGINA ZERO ED USO DELL'INTERFACCIA DI SISTEMA RS-232

$00A7-INBIT Memorizzazione transiente bit di input del ricevente.
$00A8-BITCI Conteggio bit in ingresso al ricevitore.
$00A9-RINONE Indicatore controllo bit di partenza del ricevente.
$00AA-RIDATA Locazione del byte buffer/assembly del ricevente.
$00AB-RIPRTY Memorizzazione del bit di parita' del ricevente.
$00B4-BITTS Conteggio bit di uscita dal trasmittente.
$00B5-NXTBIT Prossimo bit del trasmittente da inviare.
$00B6-RODATA Locazione byte buffer/disassembly del trasmittente.

Tutte le precedenti locazioni di pagina zero vengono usate localmente e sono state riportate come guida per la comprensione delle sottoprocedure associate. Queste non possono essere utilizzate direttamente da BASIC o da KERNAL per realizzare impieghi simili a quelli dell'RS-232. Per tali impieghi devono essere usate le sottoprocedure dell'RS-232.

LOCAZIONI DELLA MEMORIA DELLE ALTRE PAGINE ED USO DELL'INTERFACCIA DI SISTEMA RS-232

Memoria generale dell'RS-232:

$0293-M51CTR Registro di controllo pseudo 6551 (vd. fig. 6.1).
$0294-M51COR Registro di comando pseudo 6551 (vd. fig. 6.2).
$0295-M51AJB Coppia di byte che segue i registri di controllo e di comando. Queste locazioni contengono la trasmittanza relativa alla partenza del bit di controllo che ha luogo durante l'attivita' dell'interfaccia, usata alternativamente per il calcolo della tasmittanza.
$0297-RSSTAT Registro di stato dell'RS-232 (vd. fig. 6.3).
$0298-BITNUM Numero di bit trasmessi / ricevuti.
$0299-BAUDOF Coppia di byte uguagliata al tempo di una cella di bit (basata sul clock di sistema/trasmittanza).
$029B-RIDBE Indice del byte posto al termine del buffer FIFO del ricevente.
$029C-RIDBS Indice del byte posto all'inizio del buffer FIFO del

ricevente.

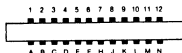
$029D-RODBS Indice del byte posto all'inizio del buffer FIFO del trasmittente.

$029E-RODBE Indice del byte posto al termine del buffer FIFO del trasmittente.

$02A1-ENABL Contiene le interruzioni di corrente attive nell'ICR del CIA #2. Quando il bit 4 e' impostato, significa che il sistema sta aspettando il fianco dell'impulso ricevente. Quando il bit 1 e' impostato, il sistema e' in fase di ricezione dati. Quando il bit 0 e' impostato, il sistema e' in fase di tasmissione dati.

PORTA UTENTE

La Porta Utente serve per collegare il COMMODORE 64 al mondo esterno. Usando le linee di collegamento disponibili per questa porta, si puo' collegare il COMMODORE 64 ad una stampante, ad un "type and talk" Votrax, ad un modem, perfino ad un altro computer.

La porta del COMMODORE 64 e' collegata direttamente ad uno dei circuiti CIA 6526. Da programma, il CIA puo' collegarsi con molti altri dispositivi.

DESCRIZIONE DEI PIN DI PORTA

PIN	DESCRIZIONE	NOTE
CIMA		
1	TERRA	
2	+5V	(Max. 100 mA)
3	RIPRISTINO	Connettendo questo pin a terra, il COMMODORE 64 esegue una PARTENZA A FREDDO, risettandosi completamente. I puntatori ad un programma BASIC vengono impostati daccapo, senza pero' che la memoria sia azzerata. Questo pin funziona anche come uscita RIPRISTINO per dispositivi esterni
4	CNT1	Contatore Porta Seriale da CIA#1 (vd. specifiche CIA)
5	SP1	Porta Seriale da CIA#1 (vd. specifiche CIA 6526)
6	CNT2	Contatore Porta Seriale da CIA#2 (vd. specifiche CIA)
7	SP2	Porta Seriale da CIA#2 (vd. specifiche CIA 6526)
8	PC2	Linea di "handshacking" da CIA#2 (vd. specifiche CIA)
9	ATN SERIALE	Pin connesso alla linea ATN del bus seriale
10	9VAC+fase	Connessi direttamente al trasformatore del COMMODORE 64 (Max. 50 mA)
11	9VAC-fase	
12	GND	
FONDO		
A	GND	Il COMMODORE 64 permette di controllare la PORTA B del circuito CIA#1, mettendo a disposizione 8 linee di input/output e 2 linee di handshacking con un dispositivo esterno.Le linee di I/O della PORTA B sono controllate da 2 locazioni, di cui una, la PORTA stessa, sita in 56577 ($DD01 HEX). Ovviamente, la lettura avviene tramite PEEK e la scrittura tramite POKE. Ogni linea di I/O puo' essere impostata sia come INPUT che come OUTPUT impostando in maniera adeguata il REGISTRO DIREZIONE DATI.
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
G	PB4	
H	PB5	
J	PB6	
K	PB7	
L	PA2	
N	GND	

Il REGISTRO DIREZIONE DATI e' allocato in 56579 ($DD03 HEX). Ciascuna delle otto linee della PORTA ha un corrispondente BIT nel REGISTRO DIREZIONE DATI a 8 bit (DDR), che controlla se la linea deve essere di input o di output. Se un bit del DDR si trova impostato a UNO, allora la corrispondente linea della PORTA e' di OUTPUT, mentre se si trova a ZERO la linea e' di INPUT. Ad esempio, se il bit 3 di DDR e' impostato a 1, allora la linea 3 della PORTA e' di OUTPUT. Un ulteriore esempio puo' essere il seguente. Supponiamo che DDR sia impostato nel modo seguente:

BIT # :	7	6	5	4	3	2	1	0
VALORE:	0	0	1	1	1	0	0	0

Si puo' vedere che le linee 5,4 e 3 sono di output poiche' i relativi bit sono a 1, mentre le rimanenti linee sono di input, dato che i relativi bit sono a 0.
Per eseguire una PEEK o una POKE sulla Porta UTENTE, e' necessario usare sia DDR che la PORTA stessa. Si ricordi che le istruzioni PEEK e POKE vogliono un numero compreso fra 0 e 255. Affinche' i numeri riportati nell'esempio possano essere usati, devono prima subire la trasformazione in decimale: tale valore e':

$$2^5 + 2^4 + 2^3 = 32 + 16 + 8 = 56$$

Da notare che il numero di bit (BIT #) per DDR e' lo stesso numero ottenuto dall'elevamento di 2 a potenza, in modo tale da ritornare il valore del bit:

(16 = 2↑4=2×2×2×2, 8 = 2↑3=2×2×2)

Le altre due linee, FLAG1 e PA2, sono diverse dal resto della PORTA UTENTE: queste due linee sono riservate esclusivamente all'"HANDSHACKING", e sono programmate diversamente dalla porta B.
L'"handshacking" e' necessario quando due dispositivi comunicano tra loro. Dato che uno dei dispositivi puo' eleborare ad una velocita' diversa da quella usata dall'altro, e' necessario far sapere a ciascun dispositivo cio' che sta facendo l'altro. L'"handshacking" e' necessario anche quando i due dispositivi lavorano alla stessa velocita', per permettere all'altro di sapere quando si deve inviare il dato, e se il dato e' stato ricevuto. La linea FLAG1 ha delle caratteristiche particolari che la rendono adatta per l'"handshacking".
FLAG1 e' un ingresso sensibile al fianco negativo di un impulso, che puo' essere usato come input ad una interruzione di carattere generale. Ogni transizione negativa sulla linea FLAG imposta il bit di interruzione di FLAG. Se viene abilitata l'interruzione FLAG, questa causa una RICHIESTA DI INTERRUZIONE. Se il bit di FLAG non e' abilitato, puo' essere interrogato dal registro interruzione sotto il controllo del programma.
PA2 e' il bit 2 della PORTA A del CIA; viene controllato come ogni altro bit della porta. Quest'ultima e' allocata nella posizione 56576 ($DD00 HEX), mentre il registro direzione dati e' allocato in 56578 ($DD02 HEX).
PER ULTERIORI INFORMAZIONI SUL 6526 SI VEDANO LE SPECIFICHE DI CIRCUITO RIPORTATE IN APPENDICE M.

BUS SERIALE

Il bus seriale e' una combinazione a "daisy chain" designata per permettere al COMMODORE 64 di comunicare con dispositivi quali il DISK DRIVE VIC-1541 e la STAMPANTE GRAFICA VIC-1525. Il vantaggio apportato dal bus seriale sta nel fatto che quest'ultimo puo' essere collegato con la porta meglio di un altro dispositivo. Al bus seriale possono essere collegati contemporaneamente non piu' di 5 dispositivi.

Il bus seriale consente tre tipi di operazione: CONTROLLO, TRASMISSIONE E RICEZIONE. Un dispositivo CONTROLLORE e' preposto alle funzioni del bus seriale; il TRASMETTITORE e' preposto all'invio di dati sul bus seriale; il RICEVITORE e' preposto alla ricezione dei dati dal bus seriale.

Il controllore del bus seriale e' il COMMODORE 64: questo si comporta anche da TRASMETTITORE (ad esempio, quando invia dati alla stampante) e da RICEVITORE (ad esempio, quando carica un programma da disco). Anche altri dispositivi possono fungere da RICEVITORE (stampante), da TRASMETTITORE (disco) o da entrambi (disco), ma solamente il COMMODORE 64 puo' comportarsi da CONTROLLORE.

Tutti i dispositivi collegati al bus seriale ricevono tutti i dati trasmessi sul bus. Per permettere al COMMODORE 64 di distribuire i dati alle proprie destinazioni, ciascun dispositivo ha un INDIRIZZO di bus, tramite il quale il COMMODORE 64 puo' controllare l'accesso al bus; gli indirizzi sul bus seriale coprono un intervallo che va da 4 a 31.

Il COMMODORE 64 puo' COMANDARE ad un particolare dispositivo di TRASMETTERE o RICEVERE. Quando ad un dispositivo viene ordinato di TRASMETTERE, questo comincia ad immettere dati sul bus seriale; quando invece gli viene ordinato di RICEVERE, il dispositivo indirizzato si trova pronto per prelevare i dati (dal COMMODORE 64 o da un altro dispositivo che si trova sul bus). Nello stesso istante, solamente un dispositivo puo' INVIARE dati sul bus, altrimenti si potrebbero verificare collisioni fra i dati, gettando il sistema nella confusione. Tuttavia, qualsiasi numero di disositivo puo' RICEVERE nello stesso istante da un TRASMETTITORE.

INDIRIZZI DEL BUS SERIALE COMUNE

NUMERO	DISPOSITIVO
4 o 5	Stampante grafica VIC-1525
8	Disk Drive VIC-1541

Sono consentiti anche altri numeri di dispositivi. Ciascun dispositivo ha il proprio indirizzo. Alcuni dispositivi (come la stampante del COMMODORE 64) forniscono all'Utente la possibilita' di scegliere fra due indirizzi.

L'INDIRIZZO SECONDARIO consente al COMMODORE 64 di trasmettere l'informazione ad un dispositivo. Ad esempio, per aprire un collegamento sul bus della stampante e per far si' che essa stampi nel formato MAIUSCOLO/MINUSCOLO, si puo' usare la seguente istruzione:

```
OPEN 1,4,7
```

dove:

1 e' il numero del file logico (il numero da usare con PRINT#),
4 e' l'INDIRIZZO della stampante, e
7 e' l'INDIRIZZO SECONDARIO che dice alla stampante di assumere il modo di stampa MAIUSCOLO/MINUSCOLO.

Per le operazioni sul bus seriale si hanno a disposizione sei linee - tre di input e tre di output. Le tre linee di input trasferiscono dati e segnali di controllo e del tempo al COMMODORE 64, le tre di output inviano dati e segnali di controllo e del tempo dal COMMODORE 64 ai dispositivi esterni posti sul bus seriale.

SPINOTTI DEL BUS SERIALE

PIN	DESCRIZIONE
1	Ingresso SRQ seriale
2	GND
3	Ingresso/Uscita ATN seriale
4	Ingresso/Uscita CLK seriale
5	Ingresso/Uscita dati seriali
6	Nessuna connessione

INGRESSO SRQ SERIALE (INGRESSO RICHIESTA DI SERVIZIO SERIALE)

Qualunque dispositivo posto sul bus seriale puo' abbassare (LOW) questo segnale quando richieda l'attenzione del COMMODORE 64; in questo caso, quest'ultimo si prende cura del dispositivo che ha causato tale abbassamento (vd. fig. 6-4).

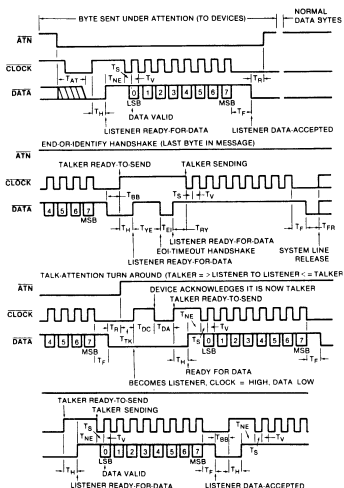



Figura 6.4 - Tempi di risposta del bus seriale

TEMPI DI RISPOSTA DEL BUS SERIALE

DESCRIZIONE	SIMBOLO	MIN	MED	MAX
Risposta ATN (richiesta) (1)	Tat	-	-	1000 ns
Predisposizione Ricevente	Th	0	-	**
Risposta non-EOI a RFD (2)	Tne	-	40 ns	200 ns
Predisposizione Trasmittente	Ts	20 ns	70 ns	-
Convalida dati	Tv	20 ns	20 ns	-
Handshake pacchetto (3)	Tf	0	20 ns	1000 ns
Pacchetto di ATN da rilasciare	Tr	20 ns	-	-
Intervallo fra due,byte	Tbb	-	-	-
Risposta di EOI	Tye	200 ns	250 ns	-
Trattenimento risposta di EOI	Tei	60 ns	-	-
Limite di risposta del trasmittente	Try	0	30 ns	60 ns
Riconoscimento del byte	Tpr	20 ns	30 ns	-

Note:

1. Se si supera il massimo tempo consentito, errore di dispositivo non presente.
2. Se si supera il massimo tempo consentito, richiesta di risposta EOI.
3. Se si supera il massimo tempo consentito, errore di pacchetto.
4. Affinche' un dispositivo esterno sia assunto come trasmittente, il valore minimo di Tv e Tpr deve essere 60 ns.

INGRESSO/USCITA ATN SERIALE (INGRESSO/USCITA ATTENZIONE SERIALE)

Il COMMODORE 64 usa questo segnale per far partire una sequenza di comandi destinati ad un dispositivo sul bus seriale. Dal momento in cui il COMMODORE 64 abbassa (LOW) il segnale, tutti i dispositivi sul bus seriale si tengono pronti a ricevere l'indirizzo trasmesso dal COMMODORE 64. Il dispositivo indirizzato deve rispondere immediatamente, altrimenti il COMMODORE 64 assume che il dispositivo chiamato non e' presente sul bus, riportando un errore nella PAROLA DI STATO (vd. fig. 6.4).

INGRESSO/USCITA CLK SERIALE (INGRESSO/USCITA CLOCK SERIALE)

Questo segnale viene usato per la temporizzazione dei dati inviati sul bus seriale (vd. fig. 6.4).

INGRESSO/USCITA DATI SERIALI

I dati presenti sul bus seriale vengono trasmessi bit per bit su questa linea (vd. fig. 6.4).

PORTA ESPANSIONE

Il connettore di espansione e' una presa femmina a 44 pin (22/22) posta sul retro del COMMODORE 64, sull'estrema destra osservando la macchina dal davanti. Il collegameno con questo connettore e' possibile usando un connettore maschio a 44 pin (22/22).

Questa porta viene usata per espansioni di sistema del COMMODORE 64 richiedenti l'accesso al bus indirizzi o al bus dati del computer. L'uso del bus espansione richiede una certa prudenza, in quanto un malfunzionamento del dispositivo allacciato a questa porta puo' causare danni al COMMODORE 64.

L'organizzazione del bus espansione e' la seguente:

I segnali disponibili sul connettore sono riportati nella tabella seguente.

NOME	PIN	DESCRIZIONE
GND	1	Terra del sistema
+5 VDC	2	(Assorbimento massimo sopportabile dai dispositivi
+5 VDC	3	PORTA UTENTE e CARTUCCIA: 450 mA)
$\overline{\text{IRQ}}$	4	Linea di RIchiesta Interruzione per 6502
R/$\overline{\text{W}}$	5	Lettura/Scrittura
DOT CLOCK	6	Timer punti video (8.18 MHz)
$\overline{\text{I/O1}}$	7	Blocco 1 di I/O ($DE00-$DEFF) non bufferizzato @
$\overline{\text{GAME}}$	8	Input ls ttl
$\overline{\text{EXROM}}$	9	Input ls ttl
$\overline{\text{I/O2}}$	10	Blocco 2 di I/O ($DE00-$DEFF) bufferizzato @ output ls ttl
$\overline{\text{ROML}}$	11	Blocco di 8K RAM/ROM decodificato ($8000) @ output ls ttl
BA	12	Segnale di bus disponibile dal circuito VIC-II non bufferizzato - carico massimo 1 ls
$\overline{\text{DMA}}$	13	Linea di richiesta di accesso diretto alla memoria input ls ttl
D7	14	Bit 7 del bus dati
D6	15	Bit 6 del bus dati
D5	16	Bit 5 del bus dati
D4	17	Bit 4 del bus dati
D3	18	Bit 3 del bus dati
D2	19	Bit 2 del bus dati
D1	20	Bit 1 del bus dati
D0	21	bit 0 del bus dati
GND	22	Terra del sistema
GND	A	
$\overline{\text{ROMH}}$	B	Blocco bufferizzato di 8K di RAM/ROM decodificati @ ($E000)
$\overline{\text{RESET}}$	C	Pin di RESET del 6502 out ttl bufferizzato/in non bufferizzato
$\overline{\text{NMI}}$	D	Interruzione non mascherabile del 6502 out ttl bufferizzato, in non bufferizzato
Ø2	E	Timer di sistema per Fase 2
A15	F	Bit 15 del bus indirizzi
A14	H	Bit 14 del bus indirizzi
A13	J	Bit 13 del bus indirizzi
A12	K	Bit 12 del bus indirizzi
A11	L	Bit 11 del bus indirizzi
A10	M	Bit 10 del bus indirizzi
A9	N	Bit 9 del bus indirizzi
A8	P	Bit 8 del bus indirizzi
A7	R	Bit 7 del bus indirizzi
A6	S	Bit 6 del bus indirizzi
A5	T	Bit 5 del bus indirizzi
A4	U	Bit 4 del bus indirizzi
A3	V	Bit 3 del bus indirizzi
A2	W	Bit 2 del bus indirizzi
A1	X	Bit 1 del bus indirizzi
A0	Y	Bit 0 del bus indirizzi
GND	Z	Terra del sistema

I nomi soprasegnati sono attivi a segnale basso

La seguente e' una descrizione di alcune linee importanti della porta espansione:

Tutte le linee sono connesse alla terra di sistema.

DOT CLOCK - Clock a punti del video a 8.18 MHz. Tutte le temporizzazioni del sistema derivano da questo orologio.

BA (Bus Available - bus disponibile) - segnale originato dal circuito VIC-II che si mantiene basso per tre cicli prima che il VIC-II acceda ai canali di trasferimento delle informazioni tra registri, rimanendo basso fino alla fine della ricerca dell'informazione da visualizzare svolta dal VIC-II.

DMA (Direct Memory Access - Accesso Diretto alla Memoria). - Quando si abbassaquesta linea, il bus indirizzi, il bus dati e la linea di lettura/scrittura del circuito processore 6510 entrano nello stato ad alta impedenza, permettendo ad un processore esterno di prendere il controllo dei canali di trasferimento delle informazioni tra registri ("busses") del sistema. Questa linea puo' essere abbassata solamente quando il clock O2 e' basso. Inoltre, poiche' il circuito VIC-II continua a visualizzare DMA, il dispositivo esterno deve essere conforme alla temporizzazione del VIC-II (vedere il diagramma di temporizzazione del VIC-II). Sul COMMODORE 64 questa linea viene tenuta alta.

CARTUCCIA CON MICROPROCESSORE Z-80

La lettura di questo libro e l'uso del COMMODORE 64 mettono in luce solamente una parte della versatilita' di questo computer. Le capacita' della macchina possono essere aumentate in maniera considerevole con l'aggiunta di strumenti periferici, quali registratori Datassette[TM], unita' disco, stampanti e modem. Tutte queste unita' possono essere aggiunte al COMMODORE 64 tramite le varie porte e prese poste sul retro della macchina. Cio' che rende valide le periferiche del COMMODORE 64 e' che queste sono "intelligenti": durante il loro funzionamento, infatti, non sfruttano lo spazio della Memoria ad Accesso Random, lasciando quindi libero l'Utente di usare tutti i 64K byte del COMMODORE 64.
Un altro vantaggio offerto dal COMMODORE 64 e' che i programmi scritti per questa macchina sono adattabili anche ad altri computer Commodore, acquistabili in futuro: parte di questo merito va alle qualita' del Sistema Operativo (OS).
Quest'ultimo, pero', non puo' rendere i programmi compatibili ad un computer prodotto da un'altra Societa'. Data comunque la semplicita' d'uso del COMMODORE 64, questa necessita' non si presentera' mai!...Ma per quegli Utenti occasionali che vogliono trarre vantaggio dal software non disponibile nello standard del COMMODORE 64, e' stata creata dalla Commodore una cartuccia CP/M [R].
Il CP/M [R] non e' un Sistema Operativo "dipendente dal computer". Al contrario, si appoggia ad una parte dello spazio di memoria normalmente disponibile per la programmazione, sfruttando questo spazio per far girare il proprio sistema operativo. Questo modo di procedere presenta vantaggi e svantaggi. Gli svantaggi sono che i programmi scritti per funzionare con il CP/M [R] devono essere piu'

corti di quelli scritti per funzionare con il sistema operativo interno del COMMODORE 64; inoltre, NON si puo' sfruttare appieno la potenza di editing dello schermo del COMMODORE 64. I vantaggi risiedono nella possibilita' di usare una grande quantita' di software creati appositamente per il CP/M [R] e per il microprocessore Z-80; inoltre, i programmi scritti usando il sistema operativo CP/M [R] possono essere trasferiti e fatti girare su qualsiasi altro computer dotato di CP/M [R] e di una scheda Z-80.

Ad esempio, la maggior parte dei computer che usano un microprocessore Z-80 richiedono che sia l'Utente stesso ad inserire nel computer la scheda Z-80. Questo metodo richiede molta attenzione a non danneggiare i delicati circuiti che attraversano il resto del computer. La cartuccia CP/M [R] della Commodore elimina questi inconvenienti, in quanto la cartuccia Z-80 viene installata sul retro del COMMODORE 64, velocemente e facilmente, senza fili confusi che in seguito potrebbero causare problemi.

COME USARE IL CP/M [R] COMMODORE

La cartuccia Z-80 della Commodore fa si' che i programmi predisposti per un microprocessore Z-80 siano in grado di funzionare sul COMMODORE 64. La cartuccia e' provvista di un dischetto contenente il sistema operativo CP/M [R] della Commodore.

ELABORAZIONE SOTTO CP/M [R] COMMODORE

Per far girare il CP/M [R]:

1) Caricare (LOAD) il programma CP/M [R] dall'unita' disco.
2) Battere RUN.
3) Premere il tasto RETURN.

A questo punto, i 64K byte di RAM del COMMODORE 64 sono accessibili al processore centrale 6510, OPPURE sono disponibili i 48K byte di RAM per il processore centrale Z-80. L'uso di questi due processori e' alternabile, ma non e' consentito il loro uso contemporaneo all'interno di un singolo programma. Cio' e' reso possibile dal sofisticato meccanismo di temporizzazione del COMMODORE 64.

La trasformazione degli indirizzi di memoria resa necessaria dall'uso della cartuccia Z-80 e' riportata nella seguente tabella. Si puo' notare che, addizionando 4096 byte alle locazioni di memoria usate in CPM [R] $1000 (HEX), si ritrovano gli indirizzi di memoria del normale sistema operativo del COMMODORE 64. La corrispondenza degli indirizzi di memoria tra Z-80 e 6510 e' la seguente:

INDIRIZZI Z-80		INDIRIZZI 6510	
DECIMALE	HEX	DEC	HEX
0000-4095	0000-0FFF	4096-8191	1000-1FFF
4096-8191	1000-1FFF	8192-12287	2000-2FFF
8192-12287	2000-2FFF	12288-16383	3000-3FFF
12288-16383	3000-3FFF	16384-20479	4000-4FFF
16384-20479	4000-4FFF	20480-24575	5000-5FFF
20480-24575	5000-5FFF	24576-28671	6000-6FFF
24576-28671	6000-6FFF	28672-32767	7000-7FFF
28672-32767	7000-7FFF	32768-36863	8000-8FFF
32768-36863	8000-8FFF	36864-40959	9000-9FFF
36864-40959	9000-9FFF	40960-45055	A000-AFFF
40960-45055	A000-AFFF	45056-49151	B000-BFFF
45056-49151	B000-BFFF	49152-53247	C000-CFFF
49152-53247	C000-CFFF	53248-57343	D000-DFFF
53248-57343	D000-DFFF	57344-61439	E000-EFFF
57344-61439	E000-EFFF	61440-65535	F000-FFFF
61440-65535	F000-FFFF	0000-4095	0000-0FFF

Per ATTIVARE la cartuccia Z-80 e DISATTIVARE il circuito 6510, digitare il seguente programma:

```
 10 REM QUESTO PROGRAMMA DEVE ESSERE USATO CON LA CARTUCCIA Z-80.
 20 REM INNANZITUTTO MEMORIZZA I DATI DELLA Z-80 NELLA LOCAZIONE $1000
 21 REM (Z80=$0000)
 30 REM POI DISATTIVA LE IRQ DEL 6510 ED ATTIVA LA CARTUCCIA Z-80.
 40 REM PER RIABILITARE IL SISTEMA 6510, OCCORRE DISABILITARE LA
 41 REM CARTUCCIA Z-80
 50 REM
 100 REM MEMORIZZA I DATI Z-80
 110 READ B: REM PRELEVA LA QUANTITA' DI CODICE Z80 CHE DEVE ESSERE
 111 REM         RIMOSSO
 120 FOR I=4096 TO 4096+B-1: REM RIMUOVE IL CODICE
 130 READ A:POKE I,A
 140 NEXT I
 200 REM LANCIA IL CODICE Z-80
 210 POKE 56333,127:REM DISATTIVA LE IRQ DEL 6510
 220 POKE 56832,00 :REM ATTIVA LA CARTUCCIA Z-80
 230 POKE 56333,129:REM ATTIVA LE IRQ DEL 6510 QUANDO SI RILASCIA
 231 REM                 LA Z-80
 240 END
 1000 REM SEZIONE DATI DEL CODICE DEL LINGUAGGIO MACCHINA Z-80
 1010 DATA 18:REM MISURA DEI DATI DA PASSARE
 1100 REM CODICE DI ATTIVAZIONE Z-80
 1110 DATA 00,00,00:REM LA CARTUCCIA Z-80 RICHIEDE DI ATTIVARE IL
 1111 REM                TEMPO A $0000
 1200 REM SEZIONE DATI Z-80 DA ELABORARE
 1210 DATA 33,02,245:REM LD  HN, NN (LOCAZIONI DELLO SCHERMO)
 1220 DATA 52:REM INCREMENTA LA LOCAZIONE HL
 1300 REM DATI Z-80 DI AUTO-ESCLUSIONE
 1310 DATA 62,01:REM LD  A,N
 1320 DATA 50,00,206:REM LD (NN),A : LOCAZIONE DI I/O
 1330 DATA 00,00,00:REM NOP:NOP:NOP
 1340 DATA 195,00,00:REM JMP $0000
```

Per maggiori dettagli sul CP/M [R] Commodore ed il microprocessore Z-80 si veda la Guida di Riferimento della cartuccia e della Z-80.

APPENDICE

APPENDICE A

ABBREVIAZIONE DELLE PAROLE CHIAVE DEL BASIC

Il BASIC del COMMODORE 64 consente di abbreviare gran parte delle parole chiave, consentendo cosi' un risparmio di tempo nella stesura dei programmi e dei comandi. L'abbreviazione di **PRINT** e' un punto interrogativo. Le abbreviazioni delle altre parole si ottengono battendo la prima o le prime due lettere della parola, seguite dalla lettera successiva tenendo contemporaneamente pigiato il tasto *SHIFT*. Se le abbreviazioni compaiono in una linea di programma, la parola chiave viene listata per intero

COM.	ABBR.	SCHERMO	COM.	ABBR.	SCHERMO	COM.	ABBR.	SCHERMO
ABS	A SHIFT B	A	LEFT$	LE SHIFT F	LE	SGN	S SHIFT G	S
AND	A SHIFT N	A	LEN	NONE	LEN	SIN	S SHIFT I	S
ASC	A SHIFT S	A♥	LET	L SHIFT E	L	SPC(	S SHIFT P	S
ATN	A SHIFT T	A	LIST	L SHIFT I	L	SQR	S SHIFT Q	S●
CHR$	C SHIFT H	C	LOAD	L SHIFT O	L	STATUS	ST	ST
CLOSE	CL SHIFT O	CL	LOG	NONE	LOG	STEP	ST SHIFT E	ST
CLR	C SHIFT L	C	MID$	M SHIFT I	M	STOP	S SHIFT T	S
CMD	C SHIFT M	C	NEW	NONE	NEW	STR$	ST SHIFT R	ST
CONT	C SHIFT O	C	NEXT	N SHIFT E	N	SYS	S SHIFT Y	S
COS	NONE	COS	NOT	N SHIFT O	N	TAB(	T SHIFT A	T♠
DATA	D SHIFT A	D♠	ON	NONE	ON	TAN	NONE	TAN
DEF	D SHIFT E	D	OPEN	O SHIFT P	O	THEN	T SHIFT H	T
DIM	D SHIFT I	D	OR	NONE	OR	TIME	TI	TI
END	E SHIFT N	E	PEEK	P SHIFT E	P	TIME$	TI$	TI$
EXP	E SHIFT X	E♣	POKE	P SHIFT O	P	USR	U SHIFT S	U♥
FN	NONE	FN	POS	NONE	POS	VAL	V SHIFT A	V♠
FOR	F SHIFT O	F	PRINT	?	?	VERIFY	V SHIFT E	V
FRE	F SHIFT R	F	PRINT#	P SHIFT R	P	WAIT	W SHIFT A	W♠
GET	G SHIFT E	G	READ	R SHIFT E	R			
GET#	NONE	GET#	REM	NONE	REM			
GOSUB	GO SHIFT S	GO♥	RESTORE	RE SHIFT S	RE♥			
GOTO	G SHIFT O	G	RETURN	RE SHIFT T	RE			
IF	NONE	IF	RIGHT$	R SHIFT I	R			
INPUT	NONE	INPUT	RND	R SHIFT N	R			
INPUT#	I SHIFT N	I	RUN	R SHIFT U	R			
INT	NONE	INT	SAVE	S SHIFT A	S♠			

APPENDICE B

CODICI DELLO SCHERMO VIDEO

La presente tabella elenca tutti i caratteri del sistema contenuti nell'insieme carattere del COMMODORE 64. Tale tabella illustra quali numeri devono essere inseriti (POKE) nella memoria dello schermo (locazioni da 1024 a 2023) per ottenere il carattere desiderato; viene anche visualizzato il carattere corrispondente ad un numero estratto (PEEK) dalla memoria dello schermo.

Sono disponibili due insiemi caratttere, di cui e' possibile adoperarne solo uno alla volta. Cio' vuol dire che non possono essere presenti sullo schermo contemporaneamente caratteri dei due insiemi. Questi insiemi vengono visualizzati premendo contemporaneamente i tasti SHIFT e C=.

Da BASIC, POKE 53272,21 seleziona il carattere maiuscolo e POKE 53272,23 quello minuscolo.

Ogni numero della tabella puo' anche essere visualizzato in campo "REVERSE": in questo caso, occorre aggiungere 128 ai valori indicati sulla tabella.

Se si vuole visualizzare un circolo pieno alla locazione 1504, occorre introdurre in tale locazione il codice del circolo: POKE 1504,81.

C'e' una locazione di memoria corrispondente al controllo del colore di ogni carattere visualizzato sullo schermo (locazioni da 55296 a 56295). Per cambiare il colore del circolo in giallo (codice colore 7) occorre introdurre il colore del carattere nella corrispondente locazione di memoria: POKE 55776,7.

Per le mappe complete del video e della memoria colore, insieme ai codici colore, si rimanda all'APPENDICE D.

CODICI DELLO SCHERMO

INS. 1	INS. 2	POKE	INS. 1	INS. 2	POKE	INS. 1	INS. 2	POKE	INS. 1	INS. 2	POKE
@		0	%		37		A	65			101
A	a	1	&		38		B	66			102
B	b	2	'		39		C	67			103
C	c	3	(		40		D	68			104
D	d	4	)		41		E	69			105
E	e	5	*		42		F	70			106
F	f	6	+		43		G	71			107
G	g	7	,		44		H	72			108
H	h	8	−		45		I	73			109
I	i	9	.		46		J	74			110
J	j	10	/		47		K	75			111
K	k	11	0		48		L	76			112
L	l	12	1		49		M	77			113
M	m	13	2		50		N	78			114
N	n	14	3		51		O	79			115
O	o	15	4		52		P	80			116
P	p	16	5		53		Q	81			117
Q	q	17	6		54		R	82			118
R	r	18	7		55		S	83			119
S	s	19	8		56		T	84			120
T	t	20	9		57		U	85			121
U	u	21	:		58		V	86		✓	122
V	v	22	;		59		W	87			123
W	w	23	<		60		X	88			124
X	x	24	=		61		Y	89			125
Y	y	25	>		62		Z	90			126
Z	z	26	?		63			91			127
[		27			64			92			
£		28						93			
]		29						94			
↑		30						95			
←		31				SPACE		96			
SPACE		32						97			
!		33						98			
"		34						99			
#		35						100			
$		36									

I codici da 128 a 255 sono le immagini reverse dei codici da 0 a 127

APPENDICE C

CODICI ASCII E CHR$

Questa Appendice illustra quali caratteri compaiono se si batte PRINT CHR$(X), per tutti i possibili valori di X. Vengono anche illustrati i valori che si ottengono battendo PRINT ASC(X), dove X e' uno qualunque dei caratteri che si possono battere. Questa funzione puo' essere utile nella valutazione di un carattere ricevuto tramite un'istruzione GET, nella conversione di un carattere da maiuscolo a minuscolo e viceversa, e nella stampa di particolari comandi che non possono essere racchiusi fra virgolette (ad es. il tasto di cambio maiuscolo/minuscolo).

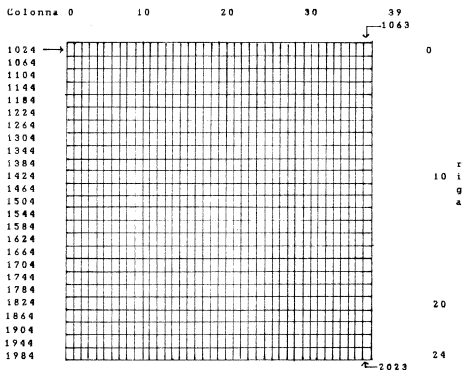
NOTA: I codici da 192 a 223 sono gli stessi di quelli da 96 a 127
         »   da 224 a 254   »   »   »   da 160 a 190
      Il codice 255   e' lo stesso del codice 126

PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$	PRINTS	CHR$
	0		26	4	52	N	78		104		129		155		180
	1		27	5	53	O	79		105		130	PUR	156		181
	2	RED	28	6	54	P	80		106		131	CRSR	157		182
	3	CRSR	29	7	55	Q	81		107		132	YEL	158		183
	4	GRN	30	8	56	R	82		108	f1	133	CYN	159		184
WHT	5	BLU	31	9	57	S	83		109	f3	134	SPACE	160		185
	6	SPACE	32	:	58	T	84		110	f5	135		161		186
	7	!	33	;	59	U	85		111	f7	136		162		187
DISABLES SHIFT C=	8	"	34	<	60	V	86		112	f2	137		163		188
ENABLES SHIFT C=	9	#	35	=	61	W	87		113	f4	138		164		189
	10	$	36	>	62	X	88		114	f6	139		165		190
	11	%	37	?	63	Y	89		115	f8	140		166		191
	12	&	38	@	64	Z	90		116	SHIFT RETURN	141		167		
RETURN	13	'	39	A	65	[	91		117	SWITCH TO UPPER CASE	142		168		
SWITCH TO LOWER CASE	14	(	40	B	66	£	92		118		143		169		
	15	)	41	C	67	]	93		119	BLK	144		170		
	16	*	42	D	68	↑	94		120	CRSR	145		171		
CRSR	17	+	43	E	69	←	95		121	RVS OFF	146		172		
RVS ON	18	,	44	F	70		96		122	CLR HOME	147		173		
CLR HOME	19	−	45	G	71		97		123	INST DEL	148		174		
INST DEL	20	.	46	H	72		98		124		149		175		
	21	/	47	I	73		99		125		150		176		
	22	0	48	J	74		100		126		151		177		
	23	1	49	K	75		101		127		152		178		
	24	2	50	L	76		102		128		153		179		
	25	3	51	M	77		103				154				

APPENDICE D

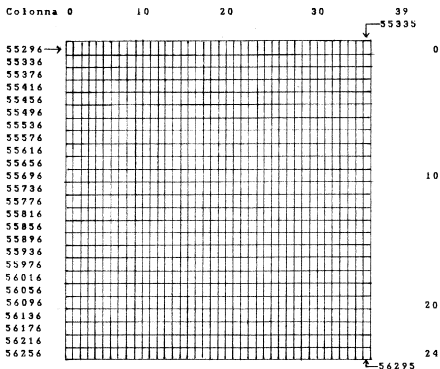
MAPPE DELLE MEMORIE SCHERMO E COLORE

La seguente tabella elenca quali locazioni di memoria controllano il posizionamento dei caratteri sullo schermo, e quali locazioni sono usate per cambiare i colori di un singolo carattere; vengono anche elencati i codici di colore disponibili per un carattere.

MAPPA DELLA MEMORIA DELLO SCHERMO

I valori attuali da introdurre (POKE) in una locazione della memoria colore, per cambiare il colore di un carattere, sono:

0	NERO	8	ARANCIO
1	BIANCO	9	MARRONE
2	ROSSO	10	ROSSO CHIARO
3	AZZURRO	11	GRIGIO 1
4	PORPORA	12	GRIGIO 2
5	VERDE	13	GRIGIO CHIARO
6	BLU	14	BLU CHIARO
7	GIALLO	15	GRIGIO 3

Ad esempio, per cambiare in rosso il colore di un carattere situato nell'angolo in alto a sinistra dello schermo, battere POKE 55296,2

MAPPA DELLA MEMORIA COLORE

APPENDICE E

VALORE DELLE NOTE MUSICALI (SCALA ANGLOSASSONE)

Questa Appendice contiene un elenco completo della nota # (diesis), della nota attuale, e dei valori da introdurre (POKE) nei registri dell'alta e bassa frequenza del circuito del suono per riprodurre la nota indicata.

NOTA MUSICALE		FREQ. OSCILLATORE			NOTA MUSICALE		FREQ. OSCILLATORE		
NOTA	OTTAVA	DEC.	ALTO	BASSO	NOTA	OTTAVA	DEC.	ALTO	BASSO
0	C-0	268	1	12	52	E-3	2703	10	143
1	C#-0	284	1	28	53	F-3	2864	11	48
2	D-0	301	1	45	54	F#-3	3034	11	218
3	D#-0	318	1	62	55	G-3	3215	12	143
4	E-0	337	1	81	56	G#-3	3406	13	78
5	F-0	358	1	102	57	A-3	3608	14	24
6	F#-0	379	1	123	58	A#-3	3823	14	239
7	G-0	401	1	145	59	B-3	4050	15	210
8	G#-0	425	1	169	64	C-4	4291	16	195
9	A-0	451	1	195	65	C#-4	4547	17	195
10	A#-0	477	1	221	66	D-4	4817	18	209
11	B-0	506	1	250	67	D#-4	5103	19	239
16	C-1	536	2	24	68	E-4	5407	21	31
17	C#-1	568	2	56	69	F-4	5728	22	96
18	D-1	602	2	90	70	F#-4	6069	23	181
19	D#-1	637	2	125	71	G-4	6430	25	30
20	E-1	675	2	163	72	G#-4	6812	26	156
21	F-1	716	2	204	73	A-4	7217	28	49
22	F#-1	758	2	246	74	A#-4	7647	29	223
23	G-1	803	3	35	75	B-4	8101	31	165
24	G#-1	851	3	83	80	C-5	8583	33	135
25	A-1	902	3	134	81	C#-5	9094	35	134
26	A#-1	955	3	187	82	C-0	9634	37	162
27	B-1	1012	3	244	83	C#-0	10207	39	223
32	C-2	1072	4	48	84	D-0	10814	42	62
33	C#-2	1136	4	112	85	F-5	11457	44	193
34	D-2	1204	4	180	86	F#-5	12139	47	107
35	D#-2	1275	4	251	87	G-5	12860	50	60
36	E-2	1351	5	71	88	G#-5	13625	53	57
37	F-2	1432	5	152	89	A-5	14435	56	99
38	F#-2	1517	5	237	90	A#-5	15294	59	190
39	G-2	1607	6	71	91	B-5	16203	63	75
40	G#-2	1703	6	167	96	C-6	17167	67	15
41	A-2	1804	7	12	97	C#-6	18188	71	12
42	A#-2	1911	7	119	98	D-6	19269	75	69
43	B-2	2025	7	233	99	D#-6	20415	79	191
48	C-3	2145	8	97	100	E-6	21629	84	125
49	C#-3	2273	8	225	101	F-6	22915	89	131
50	D-3	2408	9	104	102	F#-6	24278	94	214
51	D#-3	2551	9	247	103	G-6	25721	100	121
					104	G#-6	27251	106	115

105	A-6	28871	112	199	116	E-7	43258	168	250
106	A#-6	30588	119	124	117	F-7	45830	179	6
107	B-6	32407	126	151	118	F#-7	48556	189	172
112	C-7	34334	134	30	119	G-7	51443	200	243
113	C#-7	36376	142	24	120	G#-7	54502	212	230
114	D-7	38539	150	139	121	A-7	57743	225	143
115	D#-7	40830	159	126	122	A#-7	61176	238	248
					123	B-7	64814	253	46

IMPOSTAZIONI DEL FILTRO	
Locazione	**Contenuto**
54293	Frequenza di taglio del basso
54294	Frequenza di taglio dell'alto
54295	Risonanza (bit 4-7)
	Filtro Voce 3 (bit 2)
	Filtro Voce 2 (bit 1)
	Filtro Voce 1 (bit 0)
54296	Filtro passa alto (bit 6)
	Filtro passa banda (bit 5)
	Filtro passa basso (bit 4)
	Volume (bit 0-3)

APPENDICE F

BIBLIOGRAFIA

Addison - Wesley — "BASIC and the Personal Computer" Dwyer & Critchfield

Compute — "Compute's First Book of PET/CBM"

Cowbay Computing — "Feed me, I'm your PET Computer" Carol Alexander

"Teacher's PET - Plans, Quizzes and Answers"

Creative Computing — "Getting Acquainted With Your VIC 20" T. Hartnell

Dilithium Press — "BASIC Basic-English Dictionary for the PET" Larry Noonan

"PET BASIC" Tom Rugg & Phil Feldman

Faulk Baker Associates — "MOS Programming Manual" MOS Technology

Hayden Book Co — "BASIC from the Ground Up" David E. Simon

"I Speak BASIC to My PET" Audrey Jones, Jr.

"Library of PET subroutines" Nick Hampshire

"PET Graphics" Nick Hampshire

"BASIC Conversions Handbook, Apple, TRS 80 and PET" David A. Brain, Phillip R. Oviatt, Paul J Paquin & Chandler P. Stone

Howard W. Sams — "The Howard W. Sams Crash Course in Microcomputers" Lous E. Frenzel, Jr.

"Mostly BASIC: Application for your PET" Howard Berenbon

"PET Interfacing" James M. Downey & Steven M. Rogers

"VIC 20 Programmer's Reference Guide" A. Finkel, P. Higgingbottom, N. Harris & M. Tomzyck

Little, Brown & Co.	"Computer Games for Business, Schools and Homes" Gary W. Orwig, University of Central Florida, & William S. Hodges
Mc Graw - Hill	"Hand-On BASIC with a PET" Herbert D. Peckman
	"Home and Office Use of VisiCalc" D. Castlewitz & L. Chisauki
Osborne/Mc Graw-Hill	"PET/CBM Personal Computer Guide" Carrol S. Donahue
	"PET Fun and Games" R. Jeffries & G. Fisher
	"PET and the IEEE" A. Osborne & C. Donahue
	"Some Common BASIC Programs for the PET" L. Poole, M. Borchers & C. Donahue
	"Osborne CP/M User Guide" Thom Hogan
	"CBM Professional Computer Guide"
	"The PET Personal Guide"
	"The 8086 Book" Russel Rector & George Alexy
P. C. Publications	"Beginning Self-Teaching Computers Lessons"
Prentice - Hall	"The PET Personal Computer for Beginners" S. Dunn & V. Morgan
Reston Publishing Co.	"PET and IEEE 488 Bus (GPIB)" Eugene Fisher & C. W. Jensen
	"PET BASIC - Training Your PET Computer" Ramon Zamora, Wm. F. Carrie & B. Allbrecht
	"PET Games and Recreation" M. Ogelsby, L. Lindsley & D. Kunkin
	"PET BASIC" Richard Huskell
	"VIC Games and Recreation"
Telmas Courseware Ratings	"BASIC and the Personal Computer" T. A. Dwyer & M. Critchfield
Total Information Services	"Understanding Your PET/CBM, Vol. 1 BASIC programming"
	"Understanding Your VIC" David Shultz

Le riviste Commodore forniscono le informazioni piu' aggiornate per il COMMODORE 64. Sono disponibili su abbonamento le seguenti due pubblicazioni:

COMMODORE - Microcomputer Magazine. Pubblicazione bimestrale ($25.00 abbonamento annuale)

POWER/PLAY - The Home Computer Magazine. Pubblicazione trimestrale ($15.00 abbonamento annuale)

APPENDICE G

MAPPA DEI REGISTRI DEL CIRCUITO VIC

Register # Dec	Hex	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	S0X7							S0X0	SPRITE 0 X Component
1	1	S0Y7							S0Y0	SPRITE 0 Y Component
2	2	S1X7							S1X0	SPRITE 1 X
3	3	S1Y7							S1Y0	SPRITE 1 Y
4	4	S2X7							S2X0	SPRITE 2 X
5	5	S2Y7							S2Y0	SPRITE 2 Y
6	6	S3X7							S3X0	SPRITE 3 X
7	7	S3Y7							S3Y0	SPRITE 3 Y
8	8	S4X7							S4X0	SPRITE 4 X
9	9	S4Y7							S4Y0	SPRITE 4 Y
10	A	S5X7							S5X0	SPRITE 5 X
11	B	S5Y7							S5Y0	SPRITE 5 Y
12	C	S6X7							S6X0	SPRITE 6 X
13	D	S6Y7							S6Y0	SPRITE 6 Y
14	E	S7X7							S7X0	SPRITE 7 X Component
15	F	S7Y7							S7Y0	SPRITE 7 Y Component
16	10	S7X8	S6X8	S5X8	S4X8	S3X8	S2X8	S1X8	S0X8	MSB of X COORD.
17	11	RC8	ECM	BMM	BLNK	RSEL	YSCL2	YSCL1	YSCL0	Y SCROLL MODE
18	12	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	RASTER
19	13	LPX7							LPX0	LIGHT PEN X
20	14	LPY7							LPY0	LIGHT PEN Y
21	15	SE7							SE0	SPRITE ENABLE (ON/OFF)
22	16	N.C.	N.C.	RST	MCM	CSEL	XSCL2	XSCL1	XSCL0	X SCROLL MODE
23	17	SEXY7							SEXY0	SPRITE EXPAND Y
24	18	VS13	VS12	VS11	VS10	CB13	CB12	CB11	N.C.	SCREEN Character Memory
25	19	IRQ	N.C.	N.C.	N.C.	LPIRQ	ISSC	ISBC	RIRQ	Interupt Request's
26	1A	N.C.	N.C.	N.C.	N.C.	MLPI	MISSC	MISBC	MRIRQ	Interupt Request MASKS
27	1B	BSP7							BSP0	Background-Sprite PRIORITY
28	1C	SCM7							SCM0	MULTICOLOR SPRITE SELECT
29	1D	SEXX7							SEXX0	SPRITE EXPAND X
30	1E	SSC7							SSC0	Sprite-Sprite COLLISION
31	1F	SBC7							SBC0	Sprite-Background COLLISION

CODICI COLORE DEC HEX COLORE

32	20	0	0	BLACK	EXT 1				EXTERIOR COL
33	21	1	1	WHITE	BKGD0				
34	22	2	2	RED	BKGD1				
35	23	3	3	CYAN	BKGD2				
36	24	4	4	PURPLE	BKGD3				
37	25	5	5	GREEN	SMC 0				SPRITE MULTICOLOR 0
38	26	6	6	BLUE	SMC 1				1
39	27	7	7	YELLOW	S0COL				SPRITE 0 COLOR
40	28	8	8	ORANGE	S1COL				1
41	29	9	9	BROWN	S2COL				2
42	2A	10	A	LT RED	S3COL				3
43	2B	11	B	GRAY 1	S4COL				4
44	2C	12	C	GRAY 2	S5COL				5
45	2D	13	D	LT GREEN	S6COL				6
46	2E	14	E	LT BLUE	S7COL				7
		15	F	GRAY 3					

Legenda:
Nel modo carattere multicolore si possono usare solamente i colori da 0 a 7

APPENDICE H

FUNZIONI MATEMATICHE

Le funzioni non previste nell'insieme delle funzioni di sistema del BASIC del COMMODORE 64 possono essere calcolate come segue.

FUNZIONE	EQUIVALENTE BASIC
SECANTE	SEC(x)=1/COS(X)
COSECANTE	CSC(X)=1/COS(X)
COTANGENTE	COT(X)=1/TAN(X)
ARCOSENO	ARCSIN(X)=ATN(X/SQR(-X*X+1))
ARCOCOSENO	ARCCOS(X)=-ATN(X/SQR(-X*X+1))+p/2
ARCOSECANTE	ARCSEC(X)=ATN(X/SQR(X*X-1))
ARCOCOSECANTE	ARCCOS(X)=ATN(X/SQR(X*X-1)) +SGN(X)-1*p/2
ARCOCOTANGENTE	ARCOT(X)=ATN(X)+p/2
SENO IPERBOLICO	SINH(X)=(EXP(X)-EXP(-X))/2
COSENO IPERBOLICO	COSH(X)=(EXP(X)+EXP(-X))/2
TANGENTE IPERBOLICO	TANH(X)=EXP(-X)/EXP(X)+EXP(-X))*2+1
SECANTE IPERBOLICO	SECH(X)=2/(EXP(X)+EXP(-X))
COSECANTE IPERBOLICO	CSCH(X)=2/(EXP(X)-EXP(-X))
COTANGENTE IPERBOLICO	COTH(X)=EXP(-X)/(EXP(X)-EXP(-X))*2+1
ARCOSENO IPERBOLICO	ARCSINH(X)=LOG(X+SQR(X*X+1))
ARCOCOSENO IPERBOLICO	ARCCOSH(X)=LOG(X+SQR(X*X-1))
ARCOTANGENTE IPERBOLICO	ARCTANH(X)=LOG((1+X)/(1-X))/2
ARCOSECANTE IPERBOLICO	ARCSECH(X)=LOG(SQR(-X*X+1)+1/X)
ARCOCOSECANTE IPERBOLICO	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1/X)
ARCOCOTANGENTE IPERBOLICO	ARCCOTH(X)=LOG((X+1)/(X-1))/2

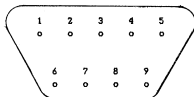
APPENDICE I

SPINOTTI DEI DISPOSITIVI DI INPUT/OUTPUT

Questa Appendice illustra le possibili connessioni realizzabili con il COMMODORE 64

1) I/O Giochi
2) Adattatore per Cartuccia
3) Audio/Video
4) I/O Seriale (Disco/Stampante)
5) Uscita Modulatore
6) Registratore a Cassette
7) Porta Utente

CONTROLLO PORTA 1		
Pin	Tipo	Note
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	PULSANTE A/LP	
7	+5V	MAX. 50mA
8	GND	
9	POT AX	

CONTROLLO PORTA 2		
Pin	Tipo	Note
1	JOYB0	
2	JOYB1	
3	JOYB2	
4	JOYB3	
5	POT BY	
6	PULSANTE B	
7	+5V	MAX. 50mA
8	GND	
9	POT BX	

CARTRIDGE EXPANSION SLOT

Pin	Type
12	BA
13	DMA
14	D7
15	D6
16	D5
17	D4
18	D3
19	D2
20	D1
21	D0
22	GND

Pin	Type
1	GND
2	+5V
3	+5V
4	IRQ
5	R/W
6	Dot Clock
7	I/O 1
8	GAME
9	EXROM
10	I/O 2
11	ROML

Pin	Type
N	A9
P	A8
R	A7
S	A6
T	A5
U	A4
V	A3
W	A2
X	A1
Y	A0
Z	GND

Pin	Type
A	GND
B	ROMH
C	RESET
D	NMI
E	S 02
F	A15
H	A14
J	A13
K	A12
L	A11
M	A10

AUDIO/VIDEO

Pin	Type	Note
1	LUMINANCE	
2	GND	
3	AUDIO OUT	
4	VIDEO OUT	
5	AUDIO IN	

SERIAL I/O

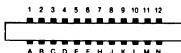
Pin	Type
1	SERIAL SRQIN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	RESET

CASSETTE

Pin	Type
A-1	GND
B-2	+5V
C-3	CASSETTE MOTOR
D-4	CASSETTE READ
E-5	CASSETTE WRITE
F-6	CASSETTE SENSE

USER I/O

Pin	Type	Note
1	GND	
2	+5V	MAX. 100 mA
3	RESET	
4	CNT1	
5	SP1	
6	CNT2	
7	SP2	
8	PC2	
9	SER. ATN IN	
10	9 VAC	MAX. 100 mA
11	9 VAC	MAX. 100 mA
12	GND	

Pin	Type	Note
A	GND	
B	FLAG2	
C	PB0	
D	PB1	
E	PB2	
F	PB3	
H	PB4	
J	PB5	
K	PB6	
L	PB7	
M	PA2	
N	GND	

APPENDICE J

CONVERSIONE DEI PROGRAMMI DAL BASIC STANDARD DEL COMMODORE 64

Se si e' in possesso di programmi scritti in un BASIC non-Commodore, puo' essere necessaria qualche modifica prima di farli "girare" sul COMMODORE 64. Quelle seguenti sono alcune note tese alla semplificazione di questa conversione.

DIMENSIONI STRINGA

Cancellare tutte le istruzioni usate per la dichiarazione della lunghezza delle stringhe. Un'istruzione come DIM A$(I,J), che dimensiona una schiera stringa di J elementi di lunghezza I, deve essere convertita nell'istruzione del BASIC Commodore DIM A$(J).
Alcuni BASIC usano per la concatenazione di stringhe una virgola o una &; entrambi devono essere cambiate con +, essendo quest'ultimo l'operatore del BASIC Commodore per la concatenazione di stringhe.
Nel BASIC del COMMODORE 64 le funzioni MID$, RIGHT$ e LEFT$ sono usate per estrarre sottostringhe da stringhe. Forme del tipo A$(I), per accedere all'I-esimo carattere, o come A$(I,J), per estrarre una sottostringa di A$ dalla posizione I alla posizione J, devono essere modificati nel modo seguente:

ALTRI BASIC	BASIC DEL COMMODORE 64
A$(I)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I-1)+X$+MID$(A$,I+1)

ASSEGNAZIONI MULTIPLE

Per impostare B e C uguali a zero, alcuni BASIC consentono istruzioni della forma:

```
10 LET B=C=0
```

Il COMMODORE 64 interpreta il secondo segno = come un operatore logico ed imposta B=-1 quando C=0; di conseguenza, un'assegnazione multipla come la precedente deve essere convertita in:

```
10 C=0 : B=0
```

ISTRUZIONI MULTIPLE

Al contrario di altri BASIC, che separano le istruzioni multiple con una barra rovescia ( ), il BASIC del COMMODORE 64 separa con due punti (:) tutte le istruzioni multiple di una linea.

FUNZIONI MAT

I programmi che usano le funzioni MAT, disponibili su alcuni BASIC, devono essere riscritti usando il ciclo FOR...NEXT.

APPENDICE K

MESSAGGI DI ERRORE

BAD DATA Da un file aperto si sono ricevuti dei dati stringa, quando il programma era in attesa di dati numerici.

BAD SUBSCIPT Tentativo del programma di fare riferimento ad un elemento di una schiera il cui numero e' al di fuori delle dimensioni specificate nell'istruzione DIM.

CAN'T CONTINUE Il comando CONT non e' in grado di funzionare, o perche' il programma non e' "girato" per il verificarsi di un errore, o perche' e' stata editata una linea.

DEVICE NOT PRESENT Il dispositivo di I/O richiesto non e' disponibile per OPEN, CLOSE, CMD, PRINT#, INPUT# o GET#

DIVISION BY ZERO La divisione per zero non e' consentita, trattandosi di un'operazione matematica non definita.

EXTRA IGNORED Sono stati battuti troppi tipi di dati in risposta ad un'istruzione INPUT. Vengono accettati solamente i tipi che corrispondono alle variabili definite nella INPUT; gli altri eccedenti, vengono persi.

FILE NOT FOUND Si e' trovato un indicatore di FINE-NASTRO durante la ricerca di un file su cassetta. Se invece si opera su disco, non c'e' nessun file con quel nome sul disco considerato.

FILE NOT OPEN Il file specificato in un'istruzione CLOSE, CMD PRINT#, INPUT# o GET# deve essere ancora aperto.

FILE OPEN Tentativo di apertura di un file usando il numero di un file gia' aperto.

FORMULA TOO COMPLEX L'espressione stringa che viene valutata deve essere divisa in almeno due parti, in modo che il sistema la possa trattare; oppure, si sono incontrate troppe parentesi nello svolgimento di una formula.

ILLEGAL DIRECT L'istruzione INPUT puo' essere usata solamente all'interno di un programma, e non in modo diretto.

ILLEGAL QUANTITY Un numero usato come argomento di una funzione di un'istruzione e' al di fuori delle dimensioni consentite.

LOAD Si sono incontrate delle difficolta' nel caricamento di un programma da nastro.

NEXT WITHOUT FOR   Errore generato: dalla non corretta nidificazione dei cicli, oppure dal nome di una variabile nell'istruzione NEXT che non corrisponde a quella dell'istruzione FOR

NOT INPUT FILE   Tentativo di lettura, tramite INPUT o GET, di un file specificato solamente "output"

NOT OUTPUT FILE   Tentativo di scrittura, tramite PRINT, di un file specificato solamente "input".

OUT OF DATA   Si e' eseguita un'istruzione READ a dati di un'istruzione DATA inesistente; oppure non ci sono altre istruzioni DATA da leggere.

OUT OF MEMORY   Non c'e' piu' memoria RAM disponibile per un programma o delle variabili; questo errore si verifica anche se ci sono troppi cicli FOR nidificati, oppure quando ci sono troppi GOSUB pendenti contemporaneamente.

OVERFLOW   Il risultato di un calcolo e' maggiore di 1.70141884E+38.

REDIM'D ARRAY   Una schiera puo' essere dimensionata solamente una volta. Se una variabile schiera viene usata prima del dimensionamento della schiera stessa, viene eseguita automaticamente un'operazione DIM su quella schiera, che imposta a dieci il numero degli elementi di quella schiera. Conseguentemente, la successiva DIM causa questo errore.

REDO FROM START   E' stato battuto un dato carattere durante un'istruzione INPUT che aspettava dati numerici. Basta ribattere il corretto valore di ingresso, ed il programma riparte automaticamente.

RETURN WITHOUT GOSUB   Si e' incontrata un'istruzione RETURN senza l'impostazione preventiva di un comando GOSUB.

STRING TOO LONG   Una stringa puo' contenere fino a 255 caratteri.

?SYNTAX ERROR   Il COMMODORE 64 non riconosce un'istruzione: c'e' una parentesi in piu' o in meno, si e' scritta male una parola chiave, ecc.

TYPE MISMATCH   Si verifica quando si usa un numero al posto di una stringa, o viceversa.

UNDEF'D FUNCTION   Si e' fatto riferimento ad una funzione definita dall'utente senza che questa fosse stata definita usando l'istruzione DEF FN.

UNDEF'D STATEMENT   Tentativo di salto (GOTO/GOSUB) o di lancio (RUN) di un numero di linea precedente

VERIFY   Il programma su nastro o su disco non corrisponde a quello correntemente in memoria.

APPENDICE L

SPECIFICHE DEL CIRCUITO MICROPROCESSORE 6510

DESCRIZIONE

Il 6510 e' un microcomputer in grado di risolvere una vasta gamma di problemi relativi a sistemi di piccola dimensione e al controllo di periferiche, di costo minimo per l'utente.
Una porta di I/O bidirezionale a 8 bit e' locata sul circuito con Registro di Uscita all'indirizzo 0000 ed il Registro Direzione Dati all'indirizzo 0001. La porta di I/O e' programmabile bit per bit.
Il bus di indirizzi a tre stati consente un accesso diretto alla memoria (Direct Memory Access, DMA), ed ai sistemi multiprocessore la divisione di una memoria comune.
L'architettura interna del Microprocessore e' identica alla Tecnologia MOS del 6502, assicurando cosi' la compatibilita' del software.

CARATTERISTICHE DEL 6510

* Porta di I/O bidirezionale a 8 bit
* Alimentazione singola a +5 Volts
* N canali, porta al silicio, tecnologia ad impoverimento di campo
* Elaborazione parallela a 8 bit
* 56 istruzioni
* Aritmetica decimale e binaria
* 13 modi di indirizzamento
* Capacita' di indicizzazione reale
* Puntatore allo stack programmabile
* Stack di lunghezza variabile
* Capacita' di interruzione
* Bus Dati bidirezionale a 8 bit
* Capacita' di accesso diretto alla memoria
* Bus compatibile con M6800
* Architettura "pipeline"
* Funzionamento a 1 e 2 Mhz
* Uso con qualunque tipo di memoria veloce

PIN CONFIGURATION
6510
1 φ1 IN
2 RDY
3 IRQ
4 NMI
5 AEC
6 VCC
7 A0
8 A1
9 A2
10 A3
11 A4
12 A5
13 A6
14 A7
15 A8
16 A9
17 A10
18 A11
19 A12
20 A13
21 GND
22 A14
23 A15
24 P5
25 P4
26 P3
27 P2
28 P1
29 P0
30 DB7
31 DB6
32 DB5
33 DB4
34 DB3
35 DB2
36 DB1
37 DB0
38 R/W
39 φ2 IN
40 RES

DATA DIRECTION REGISTER
PERIPHERAL OUTPUT REGISTER
PERIPHERAL INTERFACE BUFFER
P0 -- P7
AEC
THREE-STATE ADDRESS BUFFER
ABL
ABH
INTERNAL ADL
INTERNAL ADH
INDEX REGISTER Y
INDEX REGISTER X
STACK POINT REGISTER (S)
ALU
ACCUMULATOR A
PCL
PCH
INPUT DATA LATCH (DL)
DATA BUS BUFFER
RES IRQ NMI
INTERRUPT LOGIC
INSTRUCTION DECODE
φ2 OUT
TIMING CONTROL
φ0 IN
PROCESSOR STATUS REGISTER
R/W
INSTRUCTION REGISTER
DATA BUS
LEGEND
= 8 BIT LINE
= 1 BIT LINE

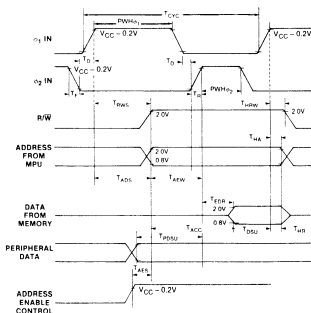
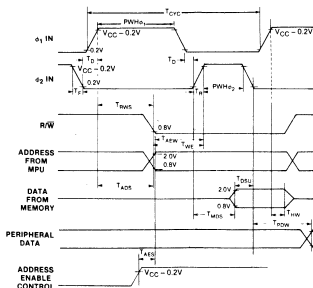
CARATTERISTICHE DEL 6510

CARATTERISTICHE DI MASSIMA			
CARATTERISTICA	SIMBOLO	VALORE	UNITÀ
Tensione di Alimentazione	Vcc	-0.3..+0.7	Vdc
Tensione di Ingresso	Vin	-0.3..+0.7	Vdc
Temperatura di Funzionamento	Ta	0..+70	Gc
Temperatura di Registrazione	Tstg	-55..+150	Gc

NOTA: Questo dispositivo contiene una protezione di ingresso contro il danneggiamento da alto voltaggio statico o campi elettrici; tuttavia, si devono prendere precauzioni per evitare l'applicazione di voltaggi superiori al massimo consentito

CARATTERISTICHE ELETTRICHE
(VCC = 5.0 V ± 5%, VSS = 0, Ta = o..+ 70 C)

CARATTERISTICHE	SIMBOLO	MIN.	TIP.	MAX.	UNITÀ
Tensione alta di ingresso:					
a) Per O1,O2	Vih	Vcc-0.2	-	Vcc+1.0	Vdc
b) Per RES, IRQ, P0-P7, Dati	"	Vss+2.0	-	-	Vdc
Tensione bassa di ingresso:					
a) Per Ø1,Ø2in	Vil	Vss-0.3	-	Vss+0.2	Vdc
b) Per RES, IRQ, P0-P7, Dati	Vil	-	-	Vss+0.8	Vdc
Perdita di corrente all'ingresso (Vin=0..5.25V, Vcc=5.25V)					
a) Logico	Iin	-	-	2.5	nA
b) Per Ø1.Ø2in	"	-	-	100	nA
Corrente di ingresso dei tre stati (OFF) (Vin=0.4..2.4V, Vcc=5.25V)					
Linee Dati	Ltsi	-	-	10	nA
Tensione alta di uscita (Ioh=-100μAdc, Vcc=4.75V)					
Dati, P0-P7, R/W, A0-A15	Voh	Vss+2.4	-	-	Vdc
Tensione bassa di uscita (Iol=1.6mAdc, Vcc=4.75V)					
Dati, P0-P7, R/W, A0-A15	Vol	-	-	Vss+0.4	Vdc
Corrente di alimentazione	Icc	-	-	125	mA
Capacita' (Vin=0, Ta=25Gc, f=1Mhz)	C				
a) Logico, P0-P7	Cin	-	-	10	pF
b) Dati		-	-	15	"
c) A0-A15, R/W	Cout	-	-	12	"
d) Ø1	CØ1	-	-	50	"
e) Ø2	CØ2	-	-	80	"

CICLO DEL CLOCK

(VCC = 5 V ± 5%, VSS = 0 V, Ta = 0..70 C)

CLOCK TIMING | **1MHz TIMING** | **2MHz TIMING**

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITS
Cycle Time	T_{CYC}	1000	—	—	500	—	—	ns
Clock Pulse Width $\phi 1$ (Measured at V_{CC} − 0.2V) $\phi 2$	PWHϕ1 PWHϕ2	430 470	— —	— —	215 235	— —	— —	ns ns
Fall Time, Rise Time (Measured from 0.2V to V_{CC} − 0.2V)	T_F, T_R	—	—	25	—	—	15	ns
Delay Time between Clocks (Measured at 0.2V)	T_D	0	—	—	0	—	—	ns

READ/WRITE TIMING (LOAD = 1TTL) | **1MHz TIMING** | **2MHz TIMING**

CHARACTERISTIC	SYMBOL	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	UNITS
Read/Write Setup Time from 6508	T_{RWS}	—	100	300	—	100	150	ns
Address Setup Time from 6508	T_{ADS}	—	100	300	—	100	150	ns
Memory Read Access Time	T_{ACC}	—	—	575	—	—	300	ns
Data Stability Time Period	T_{DSU}	100	—	—	50			ns
Data Hold Time-Read	T_{HR}		—	—				ns
Data Hold Time-Write	T_{HW}	10	30	—	10	30		ns
Data Setup Time from 6510	T_{MDS}	—	150	200	—	75	100	ns
Address Hold Time	T_{HA}	10	30	—	10	30		ns
R/W Hold Time	T_{HRW}	10	30	—	10	30		ns
Delay Time, Address valid to $\phi 2$ positive transition	T_{AEW}	180	—	—				ns
Delay Time, $\phi 2$ positive transition to Data valid on bus	T_{EDR}	—	—	395				ns
Delay Time, Data valid to $\phi 2$ negative transition	T_{DSU}	300	—	—				ns
Delay Time, R/W negative transition to $\phi 2$ positive transition	T_{WE}	130	—	—				ns
Delay Time, $\phi 2$ negative transition to Peripheral Data valid	T_{PDW}	—	—	1				μs
Peripheral Data Setup Time	T_{PDSU}	300	—	—				ns
Address Enable Setup Time	T_{AES}			60			60	ns

DESCRIZIONE DEL SEGNALE

TIMER ( )

Il 6510 richiede un timer a due fasi non sovrapposte che funziona ad un voltaggio pari a Vcc.

BUS INDIRIZZO (A0-A15)

Uscite TTL-compatibili, in grado di pilotare un carico TTL standard a 130 pF.

BUS DATI (D0-D7)

Vengono usati 8 pin. Il bus e' bidirezionale; trasferisce dati dal dispositivo alle periferiche e viceversa. Uscite: buffer a tre stati in grado di pilotare un carico TTL standard a 130 pF.

RIPRISTINO

Ingresso usato per il ripristino o l'avvio del microprocessore dopo una caduta di tensione. Durante il tempo in cui questa linea di trasmissione e' mantenuta bassa, viene inibita la scrittura da o per il microprocessore. Quando all'ingresso viene rinvenuto un margine positivo, il microprocessore attiva subito la sequenza di ripristino.
Dopo un tempo di inizializzazione del sistema pari a sei cicli del timer, viene impostato l'indicatore della maschera di interruzione, ed il microprocessore carica il contatore di programma dalle locazioni del vettore memoria FFFC e FFFD. Questa locazione e' l'inizio del controllo del programma.
Dopo che Vcc ha raggiunto 4.75V in una routine di accensione, il ripristino deve essere mantenuto basso per almeno due cicli del timer. Dopodiche' il segnale di R/W (lettura/scrittura) viene convalidato.
Quando il segnale di ripristino viene alzato a seguito di questi due cicli del temporizzatore, il microprocessore procede con la normale procedura di ripristino descritta sopra.

RICHIESTA DI INTERRUZIONE(IRQ)

Questo ingresso del livello TTL richiede l'inizio di una sequenza di interruzione nel microprocessore, che tuttavia completa l'istruzione corrente, la quale viene eseguita prima che la richiesta di interruzione venga riconosciuta. Al momento del riconoscimento, viene esaminato il bit della maschera di interruzione nel registro del Codice di Stato. Se l'indicatore della maschera di interruzione non e' impostato, il microprocessore inizia una sequenza di "interrupt". Il contatore del programma ed il Registro di Stato del processore sono registrati nello stack. Il microprocessore, allora, imposta alto l'indicatore della maschera di interruzione in modo che non possano avvenire altre interruzioni. Alla fine di questo ciclo, il contatore basso di programma viene caricato dalla locazione FFFE, quello alto dalla locazione FFFF; il controllo del programma viene percio' trasferito al vettore di memoria locato a questi indirizzi.

CONTROLLO DI ABILITAZIONE DELL'INDIRIZZO (AEC)

Il bus indirizzo e' valido solo quando il controllo di abilitazione dell'indirizzo e' alto. Se tale controllo e' basso, il bus indirizzo e' in uno stato di alta impedenza. Questa caratteristica permette facili sistemi di DMA e multiprocessore.

PORTA I/O (P0-P7)

Per la porta periferica vengono usati 8 pins, in modo da trasferire dati ai dispositivi periferici e viceversa. Il Registro di Output e' locato in RAM all'indirizzo 0001 ed il Registro Direzione Dati all'indirizzo 0000. Le uscite sono in grado di pilotare un carico TTL standard a 130 pF.

LETTURA/SCRITTURA (R/W)

Questo segnale e' generato dal microprocessore per controllare la direzione dei trasferimenti dei dati sul Bus Dati. Questa linea di trasmissione e' sempre alta, meno quando il microprocessore sta scrivendo da memoria ad un dispositivo periferico.

MODI DI INDIRIZZAMENTO

INDIRIZZAMENTO DELL'ACCUMULATORE - Forma di indirizzamento rappresentata da un'istruzione di un byte, che implica un'operazione sull'accumulatore.

INDIRIZZAMENTO IMMEDIATO - L'operando e' contenuto nel secondo byte dell'istruzione, senza che siano richiesti ulteriori indirizzamenti di memoria.

INDIRIZZAMENTO ASSOLUTO - Il secondo byte dell'istruzione specifica gli 8 bit bassi dell'indirizzo effettivo, mentre il terzo byte specifica gli 8 bit alti. Percio', il modo indirizzamento assoluto permette di accedere a tutti i 65K bytes della memoria indirizzabile.

INDIRIZZAMENTO DI PAGINA ZERO - Le istruzioni di questa pagina consentono codici e tempi di esecuzione piu' brevi, semplicemente prelevando il secondo byte dell'istruzione ed assumendo un byte di indirizzo di altezza zero. Un uso oculato della Pagina Zero puo' apportare un aumento significativo dell'efficienza dei codici.

INDIRIZZAMENTO INDICIZZATO DI PAGINA ZERO (INDIRIZZAMENTO X, Y) - Usato in congiunzione al registro indice ed indicato come "Pagina Zero, X" o "Pagina Zero, Y". L'indirizzo effettivo viene calcolato aggiungendo il secondo byte al contenuto del registro indice. Poiche' questa e' una forma dell'indirizzamento di Pagina Zero, il contenuto del secondo byte si riferisce ad una locazione in questa pagina. In aggiunta, a causa della natura di indirizzamento di Pagina Zero di questo modo, non viene sommato alcun riporto agli otto bit alti della memoria, ne' si verifica un superamento dei limiti di pagina.

INDIRIZZAMENTO INDICIZZATO ASSOLUTO (INDIRIZZAMENTO X, Y) - Usato in congiunzione ai registri indice X ed Y ed indicato come "Assoluto, X" e "Assoluto, Y". L'indirizzo effettivo e' formato sommando i contenuti

X e Y all'indirizzo contenuto nel secondo e terzo byte dell'istruzione. Questo modo consente al registro indice di contenere l'indirizzo o valore di conteggio, ed all'istruzione di contenere l'indirizzo base. Questo tipo di indirizzamento consente inoltre di fare riferimento a qualunque locazione, ed all'indice di modificare campi multipli, il che comporta una riduzione della codifica e del tempo di esecuzione.

INDIRIZZAMENTO IMPLICITO - L'indirizzo contenenente l'operando e' fissato implicitamente nel codice operativo dell'istruzione.

INDIRIZZAMENTO RELATIVO - Usato solamente con istruzioni di salto per stabilire la destinazione di un salto condizionato.
Il secondo byte dell'istruzione diviene l'operando, che, sotto forma di "offset", viene sommato al contenuto degli 8 bit bassi del contatore di programma quando tale contatore viene impostato alla prossima istruzione. Le dimensioni dell'offset vanno da -128 a +128 bytes a partire dalla prossima istruzione.

INDIRIZZAMENTO INDICIZZATO INDIRETTO (INDIRETTO,X) - Il secondo byte dell'istruzione e' sommato al contenuto del registro indice X, tralasciando il riporto. Il risultato di questa addizione punta ad una locazione di memoria a Pagina Zero il cui contenuto e' costituito dagli 8 bit alti dell'indirizzo effettivo. Entrambi le locazioni di memoria che specificano i bytes alto e basso dell'indirizzo effettivo devono essere a Pagina Zero.

INDIRIZZAMENTO INDIRETTO INDICIZZATO (INDIRETTO, Y) - Il secondo byte dell'istruzione punta ad una locazione di memoria a Pagina Zero. Il contenuto di questa locazione di memoria viene sommato a quello del registro indice Y; il risultato rappresenta gli 8 bit bassi dell'indirizzo effettivo. Il riporto di questa addizione e' sommato al contenuto della successiva locazione di memoria di Pagina Zero; il risultato rappresenta gli 8 bit alti dell'indirizzo effettivo.

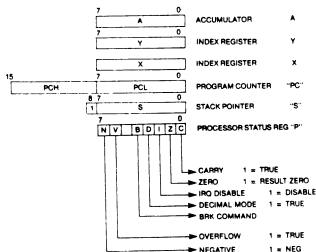
INDIRIZZAMENTO INDIRETTO ASSOLUTO - Il secondo byte dell'istruzione contiene gli 8 bit bassi della locazione di memoria; gli 8 bit alti di quella locazione sono contenuti nel terzo byte dell'istruzione. Il contenuto della locazione interamente determinata della memoria e' il byte basso dell'indirizzo effettivo. La locazione di memoria successiva contiene il byte alto dell'indirizzo effettivo, che viene caricato nei 16 bit del contatore di programma.

INSIEME DELLE ISTRUZIONI

SEQUENZA ALFABETICA

ADC	Somma la Memoria all'Accumulatore, con Riporto
AND	"AND" fra Memoria ed Accumulatore
ASL	Scorrimento (Shift) a Sinista di un bit (Memoria o Accumulatore)
BCC	Salto sull'azzeramento del Riporto
BCS	Salto sull'impostazione del Riporto
BEQ	Salto su Risultato Zero
BIT	Confronta i bit nella Memoria con l'Accumulatore
BMI	Salto su Risultato Meno
BNE	Salto su Risultato Non-Zero
BPL	Salto su Risultato Piu'
BRK	Interruzione (break) forzata
BVC	Salto sull'Azzeramento dell'Overflow
BVS	Salto sull'Impostazione dell'Overflow
CLC	Azzera l'Indicatore di Riporto
CLD	Azzera il Modo Decimale
CLI	Azzera l'Interruzione e Disabilita il Bit
CLV	Azzera l'Indicatore di Overflow
CMP	Compara Memoria ed Accumulatore
CPX	Compara Memoria ed Indice X
CPY	Compara Memoria ed Indice Y
DEC	Decrementa la Memoria di uno
DEX	Decrementa l'Indice X di uno
DEY	Decrementa l'Indice Y di uno
EOR	OR esclusivo della Memoria con l'Accumulatore
INC	Incrementa la Memoria di uno
INX	Incrementa l'Indice X di uno
INY	Incrementa l'Indice Y di uno
JMP	Salto a Nuova Locazione
JSR	Salto a Nuova Locazione e salvataggio dell'indirizzo di ritorno
LDA	Carica l'Accumulatore con la Memoria
LDX	Carica l'Indice X con la Memoria
LDY	Carica l'Indice Y con la Memoria
LSR	Scorrimento a Destra (Shift) di un Bit (Memoria o Accumulatore)
NOP	Nessuna Operazione
ORA	OR della Memoria con l'Accumulatore
PHA	Posiziona l'Accumulatore sullo Stack
PHP	Posiziona lo Stato del Processore sullo Stack

PLA	Ritira l'accumulatore dallo Stack
PLP	Ritira lo Stato del Processore dallo Stack
ROL	Ruota a Sinitra di un Bit (Memoria o Accumulatore)
ROR	Ruota a Destra di un Bit (Memoria o Accumulatore)
RTI	Ritorno da un'Interruzione
RTS	Ritorno da una Subroutine
SBC	Sottrae la Memoria dall'Accumulatore, con Prestito
SEC	Imposta l'Indicatore di Riporto
SED	Imposta il Modo Decimale
SEI	Imposta lo Stato di Disabilitazione dell'interruzione
STA	Registra l'Accumulatore in Memoria
STX	Registra l'Indice X in Memoria
STY	Registra l'Indice Y in Memoria
TAX	Trasferisce l'Accumulatore all'Indice X
TAY	Trasferisce l'Accumulatore all'Indice Y
TSX	Trasferisce il Puntatore allo Stack all'Indice X
TXA	Trasferisce l'Indice X all'Accumulatore
TXS	Trasferisce l'Indice X al Registro dello Stack
TYA	Trasferisce l'Indice Y all'Accumulatore

MODELLO DI PROGRAMMAZIONE

INSIEME ISTRUZIONI - CODICI OPERATIVI, TEMPI DI ESECUZIONE, REQUISITI DELLA MEMORIA

INSTRUCTIONS		Immediate			Absolute			Zero Page			Accum			Implied			(Ind.) X			(Ind.) Y			Z Page, X			Abs, X			Abs, Y			Relative			Indirect			Z Page, Y			CONDITION CODES
Mnemonic	Operation	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N Z C I D V
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										✓ ✓ ✓ — — ✓
AND	A ∧ M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										✓ ✓ — — — —
ASL	C ← [7 … 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													✓ ✓ ✓ — — —
BCC	BRANCH ON C = 0 (2)																															90	2	2							— — — — — —
BCS	BRANCH ON C = 1 (2)																															B0	2	2							— — — — — —
BEQ	BRANCH ON Z = 1 (2)																															F0	2	2							— — — — — —
BIT	A ∧ M				2C	4	3	24	3	2																															M_7 ✓ — — — M_6
BMI	BRANCH ON N = 1 (2)																															30	2	2							— — — — — —
BNE	BRANCH ON Z = 0 (2)																															D0	2	2							— — — — — —
BPL	BRANCH ON N = 0 (2)																															10	2	2							— — — — — —
BRK	(See Fig. 1)													00	7	1																									— — — 1 — —
BVC	BRANCH ON V = 0 (2)																															50	2	2							— — — — — —
BVS	BRANCH ON V = 1 (2)																															70	2	2							— — — — — —
CLC	0 → C													18	2	1																									— — 0 — — —
CLD	0 → D													D8	2	1																									— — — — 0 —
CLI	0 → I													58	2	1																									— — — 0 — —
CLV	0 → V													B8	2	1																									— — — — — 0
CMP	A − M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										✓ ✓ ✓ — — —
CPX	X − M	E0	2	2	EC	4	3	E4	3	2																															✓ ✓ ✓ — — —
CPY	Y − M	C0	2	2	CC	4	3	C4	3	2																															✓ ✓ ✓ — — —
DEC	M − 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													✓ ✓ — — — —
DEX	X − 1 → X													CA	2	1																									✓ ✓ — — — —
DEY	Y − 1 → Y													B8	2	1																									✓ ✓ — — — —
EOR	A ⊻ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										✓ ✓ — — — —
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													✓ ✓ — — — —
INX	X + 1 → X													E8	2	1																									✓ ✓ — — — —
INY	Y + 1 → Y													C8	2	1																									✓ ✓ — — — —
JMP	JUMP TO NEW LOC				4C	3	3																												6C	5	3				— — — — — —
JSR	(See Fig. 2) JUMP SUB				20	6	3																																		— — — — — —
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										✓ ✓ — — — —

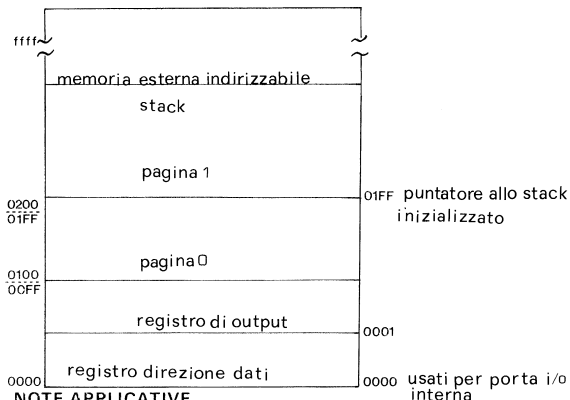
INSTRUCTIONS		Immediate			Absolute			Zero Page			Accum.			Implied			(Ind.) X			(Ind.) Y			Z. Page, X			Abs. X			Abs. Y			Relative			Indirect			Z. Page, Y			CONDITION CODES
Mnemonic	Operation	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N Z C I D V
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3							B6	4	2	✓ ✓ — — — —
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													B4	4	2	BC	4	3													✓ ✓ — — — —
LSR	0 → [7 … 0] → C				4E	6	3	46	5	2	4A	2	1										56	6	2	5E	7	3													0 ✓ ✓ — — —
NOP	NO OPERATION													EA	2	1																									— — — — — —
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3										✓ ✓ — — — —
PHA	A → M_S S − 1 → S													48	3	1																									— — — — — —
PHP	P → M_S S − 1 → S													08	3	1																									— — — — — —
PLA	S + 1 → S M_S → A													68	4	1																									✓ ✓ — — — —
PLP	S + 1 → S M_S → P													28	4	1																									(RESTORED)
ROL	← [7 … 0] ← [C] ←				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3													✓ ✓ ✓ — — —
ROR	→ [C] → [7 … 0] →				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3													✓ ✓ ✓ — — —
RTI	(See Fig. 1) RTRN INT													40	6	1																									(RESTORED)
RTS	(See Fig. 2) RTRN SUB													60	6	1																									— — — — — —
SBC	A − M − C̄ → A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3										✓ ✓ (3) — — ✓
SEC	1 → C													38	2	1																									— — 1 — — —
SED	1 → D													F8	2	1																									— — — — 1 —
SEI	1 → I													78	2	1																									— — — 1 — —
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3										— — — — — —
STX	X → M				8E	4	3	86	3	2																												96	4	2	— — — — — —
STY	Y → M				8C	4	3	84	3	2													94	4	2																— — — — — —
TAX	A → X													AA	2	1																									✓ ✓ — — — —
TAY	A → Y													A8	2	1																									✓ ✓ — — — —
TSX	S → X													BA	2	1																									✓ ✓ — — — —
TXA	X → A													8A	2	1																									✓ ✓ — — — —
TXS	X → S													9A	2	1																									— — — — — —
TYA	Y → A													98	2	1																									✓ ✓ — — — —

(1) ADD 1 TO N IF PAGE BOUNDARY IS CROSSED
(2) ADD 1 TO N IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO N IF BRANCH OCCURS TO DIFFERENT PAGE
(3) CARRY NOT = BORROW
(4) IF IN DECIMAL MODE Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X INDEX X
Y INDEX Y
A ACCUMULATOR
M MEMORY PER EFFECTIVE ADDRESS
M_S MEMORY PER STACK POINTER

+ ADD
− SUBTRACT
∧ AND
∨ OR
⊻ EXCLUSIVE OR

✓ MODIFIED
— NOT MODIFIED
M_7 MEMORY BIT 7
M_6 MEMORY BIT 6
N NO CYCLES
NO BYTES

NOTA: Il COMMODORE SEMICONDUCTOR GROUP non si assume responsabilita' sull'uso di CODICI OPERATIVI non definiti

NOTE APPLICATIVE

Locando il Registro di Uscita alla Porta di I/O interno a Pagina Zero, si migliora il rendimento delle istruzioni di indirizzamento di Pagina Zero del 6510.

Assegnando ai pin di I/O i valori necessari per la predisposizione ad ingresso (usando il Registro di Direzione dei Dato), viene data all'Utente la possibilita' do cambiare il contenuto dell'indirizzo 0001 (Registro di Uscita) usando dispositivi periferici. L'unione di queste due caratteristiche consente la creazione di nuove e versatili tecniche di programmazione, non realizzabili precedentemente.

APPENDICE M

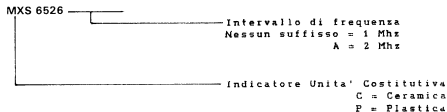
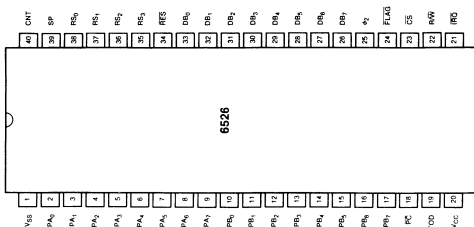
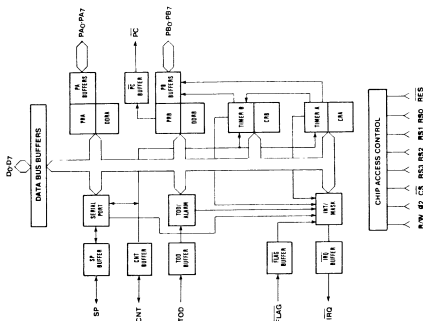
SPECIFICHE DEL CIRCUITO ADATTATORE INTERFACCIA COMPLESSA 6526 (CIA)

DESCRIZIONE

L'Adattatore Interfaccia Complessa CIA 6526 e' un dispositivo interfaccia periferica, compatibile al Bus 65XX, dotato di un timer e di capacita' di I/O estremamente flessibili.

CARATTERISTICHE

* 16 linee di trasmissione I/O programmabili separatamente
* "Handshacking" a 8 o 16 bit per lettura/scrittura
* 2 timer di intervalli a 16 bit, indipendenti e collegabili
* Orologio a 24 ore (AM/PM) con allarme programmabile
* Registro di scorrimento a 8 bit per I/O seriale
* Capacita' di carico di 2 TTL
* Linee di trasmissione I/O CMOS-compatibili
* Disponibilita' operativa a 1 o 2 Mhz

PREDISPOSIZIONE

VALORI MASSIMI

Alimentazione (Vcc)	-0.3V...+7.0V
Tensione di Ingresso/Uscita (Vin)	-0.3V...+7.0V
Temperatura di funzionamento (Top)	0...70 C
Temperatura di registrazione (Tstg)	-55...+150 C

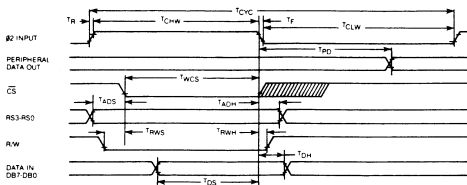
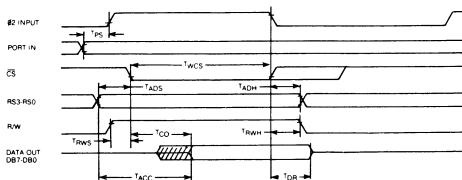
Tutti gli ingressi sono dotati di un circuito di protezione da cariche elettrostatiche; e' consigliabile evitare l'applicazione non necessaria di voltaggi superiori ai limiti di tolleranza.

COMMENTO

Tensioni superiori a quelle sopra elencate possono arrecare danni irreparabili al dispositivo. Non e' implicita l'operativita' funzionale di quest'ultimo, in condizioni analoghe o superiori a quelle indicate nelle sezioni operative di questa Specifica, e l'esposizione per lunghi periodi a condizioni di massima tensione puo' comprometterne l'affidabilita'.

CARATTERISTICHE ELETTRICHE
(VCC ± 5%, VSS = 0 V, Ta = 0..70GC)

CARATTERISTICHE	SIMBOLO	MIN.	TIP	MAX	UNITÀ
Tensione alta di ingresso	Vih	+2.4	-	Vcc	V
Tensione bassa di ingresso	Vil	-0.3	-	-	V
Perdita di tensione all'ingresso- Vin=Vss+5V (TOD,R/W,FLAG,CS O2,RES,RS0-RS3)	Iin	-	1.0	2.5	nA
Resistenza pull-up della Porta Ingresso	Rpi	3.1	5.0	-	KOhm
Perdita di tensione all'uscita per Stato ad alta impedenza (Tre Stati) Vin=4...2.4V (DB0-DB7,SP,CNT,IRQ)	ltsi	-	+-1.0	+-10.0	nA
Tensione alta di uscita Vcc=MIN,Iload < -200nA (PA0-PA7,PC,PB0-PB7,DB0-DB7)	Voh	+2.4	-	Vcc	V
Tensione bassa di uscita Vcc=MIN, Iload < 3.2mA	Vol	-	-	+0.40	V
Tensione alta di uscita (sourcing) Voh > 2.4V (PA0-PA7,PB0-PB7,PC,DB0-DB7)	Ioh	-200	-1000	-	nA
Corrente bassa di uscita (sinking) Vol < 0.4V (PA0-PA7,PC,PB0-PB7,DB0-DB7)	Iol	3.2	-	-	mA
Capacita' ingresso	Cin	-	7	10	pF
Capacita' uscita	Cout	-	7	10	pF
Corrente di alimentazione	Icc	-	70	100	mA

TEMPO LETTURA

TEMPO SCRITTURA

SEGNALI DELL'INTERFACCIA 6526

02 – INGRESSO TIMER

Ingresso TTL-compatibile usato per operazioni interne al dispoqitivo, e come riferimento temporale per la comunicazione con il bus dati del sistema.

CS – INGRESSO SELEZIONE CIRCUITO

Controlla l'attivita' del 6526. Un basso livello su CS mentre O2 e' alto causa la risposta del dispositivo a segnali provenienti dalle linee di lettura-scrittura e degli indirizzi. Se invece CS e' alto, viene evitato a queste linee il controllo del 6526. La linea CS e' attivata normalmente (bassa) a O2 da un'apposita combinazione dell'indirizzo.

R/W – INGRESSO LETTURA/SCRITTURA

Segnale fornito normalmente dal microprocessore; controlla la direzione del trasferimento dei@dati del 6526. Se R/W e' alto si ha una lettura (trasferimento fuori dal 6526), se invece e' basso si ha una scrittura (trasferimento@dentro al 6526).

RS3-RSO – INGRESSI INDIRIZZO RS3-RSO

Selezionano i registri interni come descritto dalla mappa registri

DB7-DBO – INGRESSI/USCITE BUS DATI

Gli 8 pin del Bus Dati trasferiscono le informazioni tra il 6526 ed il Bus Dati del sistema. Finche' CS e' basso e R/W e O2 sono alti, in modo da leggere dal dispositivo, questi pin sono uscite ad alta impedenza. Durante la lettura, vengono abilitati i buffer di@uscita del Bus Dati, che DIRIGE tali dati dal registro selezionato sul Bus Dati del sistema.

IRQ – USCITA RICHIESTA DI INTERRUZIONE

Uscita a condotto aperto connessa normalmente all'ingresso di interruzione del processore. Un resistore esterno in conduzione mantiene il segnale alto, permettendo l'interconnessione di uscite IRQ multiple. L'uscita IRQ e' normalmente spenta (alta impedenza) ed e' attivata bassa nel modo indicato nella descrizione funzionale.

RES – INGRESSO RIPRISTINO

Se il pin RES e' basso, vengono azzerati tutti i registri interni. I pin di porta sono impostati a zero come i registri e gli@ingressi di porta (anche se una lettura delle porte li imposta tutti alti@a causa di conduzioni passive). I registri di controllo del timer sono impostati a zero; il timer li imposta tutti@a uno, salvandoli in un registro tampone e ripristinando tutti gli altri a zero.

CARATTERISTICHE DI SINCRONIZZAZIONE DEL 6526

CARATTERISTICA	SIMBOLO	1 MHz		2MHz		UNITÀ
		MIN	MAX	MIN	MAX	
CLOCK 02						
Tempo del ciclo	Tcyc	1000	20000	500	20000	ns
Tempo di salita e discesa	Tr,Tf	–	25	–	25	ns
Ampiezza di pulsazione del clock (alto)	Tchw	420	10000	200	10000	ns
Ampiezza di pulsazione del clock (basso)	Tclw	420	10000	200	10000	ns
CICLO DI SCRITTURA						
Ritardo di uscita da 02	Tpd	–	1000	–	500	ns
CS basso mentre 02 alto	Twcs	420	1000	200	–	ns
Tempo preparazione ind	Tads	0	–	0	–	ns
Tempo conservazione ind	Tadh	10	–	5	–	ns
Tempo preparazione R/W	Trws	0	–	0	–	ns
Tempo conservazione R/W	Trwh	0	–	0	–	ns
Tempo preparazione Bus Dati	Tds	150	–	75	–	ns
Tempo conservazione Bus Dati	Tdh	0	–	70	–	ns
CICLO DI LETTURA						
Tempo preparazione Porta	Tps	300	–	150	–	ns
CS basso mentre 02 alto (2)	Twcs	420	–	20	–	ns
Tempo preparazione ind	Tads	0	–	0	–	ns
Tempo conservazione ind	Tadh	10	–	5	–	ns
Tempo preparazione R/W	Trws	0	–	0	–	ns
Tempo conservazione R/W	Trwh	0	–	0	–	ns
Accesso ai Dati						
Accesso ai Dati da RS3-RS0	Tacc	–	550	–	275	ns
Accesso ai Dati da CS (3)	Tco	–	320	–	150	ns
Tempo rilascio						
Tempo rilascio dei Dati	Tdr	50	–	25	–	ns

NOTE:

1) Tutti i tempi sono riferiti a Vil max e Vih min per gli ingressi, ed a Vol max e Voh min per le uscite.
2) Twcs e' misurato dal piu' recente fra 02 alto e CS basso.
3) Tco e' misurato dal piu' recente fra Or alto e CS basso. Dati validi sono a disposizione solo dopo il piu' recente fra Tacc e Tco.

MAPPA DEI REGISTRI

RS3	RS2	RS1	RS0	REG	NOME	
0	0	0	0	0	PRA	Registro Dati Periferici A
0	0	0	1	1	PRB	Registro Dati Periferici B
0	0	1	0	2	DDRA	Registro Direzione Dati A
0	0	1	1	3	DDRB	Registro Direzione Dati B
0	1	0	0	4	TA LO	Registro basso del Timer A
0	1	0	1	5	TA HI	Registro alto del Timer A
0	1	1	0	6	TB LO	Registro basso del Timer B
0	1	1	1	7	TB HI	Registro alto del Timer B
1	0	0	0	8	TOD10TH	Registro Decimi di Secondo
1	0	0	1	9	TOD SEC	Registro dei Secondi
1	0	1	0	A	TOD MIN	Registro dei Minuti
1	0	1	1	B	TOD HR	Registro delle Ore
1	1	0	0	C	SDR	Registro Dati Seriali
1	1	0	1	D	ICR	Reg. Controllo Interruz.
1	1	1	0	E	CRA	Registro di Controllo A
1	1	1	1	F	CRB	Registro di Controllo B

DESCRIZIONE FUNZIONALE DEL 6526

PORTE DI I/O (PRA, PRB, DDRA, DDRB)

Le porte A e B consistono ognuna di un Registro Dati Periferici (PR) a 8 bit, e di un Registro Direzione Dati (DDR), anch'esso a 8 bit. Se un bit del DDR e' impostato a 1, il corrispondente bit del PR e' impostato ad uscita; se un bit del DDR e' zero, il corrispndente bit del PR e' impostato ad ingresso. Su una LETTURA, il PR riflette l'informazione presente sugli attuali pin di porta (PA0-PA7, PB0-PB7) sia per i bit di ingresso che per quelli di uscita. Le porte A e B montano dei dispositivi sia in conduzione passiva che in conduzione attiva, fornendo cosi' la compatibilita' sia con CMOS che con TTL. Oltre alle normali operazioni di I/O, PB6 e PB7 forniscono anche funzioni di uscita al timer.

«HANDSHACKING»

L'"handshacking" sul trasferimento dati puo' essere realizzato usando il pin di uscita PC ed il pin di ingresso FLAG. PC rimane basso per un ciclo, in modo da seguire una lettura o una scrittura della PORTA B. Questo segnale puo' essere usato per indicare "dati disponibili" alla PORTA B o "dati accettati" dalla PORTA B.L'"handshacking" su trasferimenti di dati a 16 bit (usando la PORTA A e la PORTA B) puo' essere eseguito leggendo o scrivendo sempre la PORTA A per prima. FLAG e' un ingresso sensibile al fianco negativo di un impulso; puo' essere usato per la ricezione dell'uscita di un altro 6526, o come ingresso di un interruttore di uso generale. Qualunque transizione negativa di FLAG imposta il bit di interruzione di FLAG.

REG	NOME	D7	D6	D5	D4	D3	D2	D1	D0
0	PRA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	PRB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
2	DDRA	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
3	DDRB	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0

TIMER DI INTERVALLO (TIMER A, TIMER B)

Ogni timer di intervallo e' formato da un contatore del Timer a sola lettura a 16 bit , e da un "latch" del Timer a sola scrittura a 16 bit. I dati scritti per il timer sono trattenuti nel "latch" del timer stesso, mentre i dati letti da quest'ultimo costituiscono il contenuto attuale del Contatore del Timer. I Timer possono essere usati singolarmente o in collegamento per operazioni estese. I vari modi del Timer consentono la generazione di lunghi tempi di ritardo, pulsazioni di ampiezza variabile, treni di pulsazioni e forme d'onda di frequenza variabile. Utilizzando l'ingresso CNT, i Timer sono in grado di contare pulsazioni esterne, oppure misurare frequenza, ampiezza di pulsazione e tempi di ritardo di segnali esterni. Ciascun Timer ha un registro di controllo associato, che fornisce un controllo indipendente delle seguenti funzioni:

* START/STOP

Un bit di controllo permette al microprocessore di avviare o arrestare il Timer ad ogni istante

* PB ON/OFF

Un bit di controllo permette all'uscita del Timer di apparire su una linea di uscita della PORTA B (PB6 per il Timer A e PB7 per il Timer B). Questa funzione si sovrappone al bit di controllo del DDRB, forzando ad uscita la linea di PB adatta.

* BISTABILE/PULSAZIONE

Un bit di controllo seleziona l'uscita applicata alla PORTA B. Quando si verifica un "underflow" in ogni Timer, l'uscita puo' entrare in una condizione bistabile, oppure generare una singola pulsazione positiva della durata di un ciclo. L'uscita bistabile e' impostata alta tutte le volte che il timer viene avviato ed impostato basso da RES.

* MONOSTABILE/CONTINUO

Un bit di controllo seleziona tutti e due i modi del Timer. Nel modo monostabile, il Timer esegue un conteggio alla rovescia dal valore trattenuto a zero, genera un'interruzione, ricarica il valore trattenuto e ripete la procedura, continuamente.

* CARICAMENTO FORZATO

Un bit selettore permette al "latch" del Timer di essere caricato ad ogni istante nel contatore del Timer, che quest'ultimo sia in funzione o no

* MODO INGRESSO

Un gruppo di bit di controllo consente la selezione del clock usato per decrementare il Timer. Il TIMER A puo' contare 02 segnali di temporizzazione, oppure pulsazioni esterne applicate al pin CNT. Il TIMER B puo' contare 02 pulsazioni, pulsazioni esterne del CNT, pulsazioni di "underflow" del Timer A o pulsazioni di "underflow" del Timer A mentre il pin CNT viene tenuto alto.
Il "latch" del Timer viene caricato nel Timer stesso al verificarsi di qualunque "underflow" di tale Timer, in seguito ad un caricamento forzato o ad una scrittura sul byte alto del "prescaler" mentre il Timer e' fermo. Se quest'ultimo e' avviato, una scrittura sul byte alto carica il latch del Timer, ma non ricarica il contatore.

READ (TIMER)

REG	NOME								
4	TA LO	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
5	TA HI	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
6	TB LO	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
7	TB HI	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0

WRITE (PRESCALER)

REG	NOME								
4	TA LO	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PAL1	PAL0
5	TA HI	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAH0
6	TB LO	PBL7	PBL6	PBL5	PBL4	PBL3	PBL2	PBL1	PBL0
7	TB HI	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0

CLOCK TEMPO DEL GIORNO (TOD)

Il clock TOD e' un timer di uso particolare destinato ad applicazioni in tempo reale. Consiste in un orologio di 24 ore (AM/PM) con risoluzione di 1/10 di secondo. E' organizzato su quattro registri: decimi di secondo, secondi, minuti ed ore. L'indicatore AM/PM si trova nel MSB (Most Significant Byte-byte piu' significativo) del registro delle ore, facilitando il controllo dei bit. Ogni registro esegue la lettura in formato BCD (BInary Coded Decimal) allo scopo di agevolare la conversione per il pilotaggio di video, ecc. Il clock richiede un ingresso esterno di 60 Hz o 50 Hz (programmabili) a livello di TTL, sul pin di TOD, per un accurato controllo del tempo. Inoltre, viene fornito un'allarme programmabile per la generazione di un'interruzione all'istante desiderato. I registri dell'ALLARME sono locati agli stessi indirizzi dei corrispondenti registri del TOD. L'accesso all'ALLARME e' governato da un bit del Registro di Controllo. ALLARME risiede in una memoria a sola scrittura; qualsiasi lettura di un indirizzo TOD provoca la lettura del tempo senza curarsi dello stato del bit di accesso ad ALLARME.
Per un'appropriata impostazione e lettura del TOD, si deve seguire una particolare sequenza di eventi. Non appena si verifica una scrittura sul registro delle Ore, il TOD si ferma automaticamente, ed il clock non riparte fino a dopo che si e' verificata una scrittura sul registro dei decimi di secondo. Cio' assicura sempre la partenza del TOD all'istante desiderato. Poiche' ad ogni istante, rispetto ad un'operazione di lettura, puo' verificarsi un riporto da uno stadio

all'altro, e' compresa una funzione di intrappolaento nel "latch" che mantenga costanti tutte le informazioni del TOD durante una sequenza di lettura. Ad una lettura delle ore, tutti e quattro i registri del TOD vengono intrappolati nel "latch", rimanendo in questo stato fino a dopo che si e' verificata una scrittura sul registro dei decimi di secondo. Quando i registri di uscita sono intrappolati in un circuito "latch", il clock del TOD continua a contare. Se deve essere letto un solo registro, non ci sono problemi di riporto ed il registro puo' essere letto "al volo", a patto che ogni lettura delle ore sia seguita da una lettura dei decimi di secondo, in modo da disabilitare l'intrappolamento sul circuito "latch".

READ

REG	NOME								
8	TOD10THS	0	0	0	0	T8	T4	T2	T1
9	TOD SEC	0	SH4	SH2	SH1	SL8	SL4	SL2	SL1
A	TOD MIN	0	MH4	MH2	MH1	ML8	ML4	ML2	ML1
B	TOD HR	PM	0	0	HH	HL8	HL4	HL2	HL1

WRITE

CRB7=0 TOD
CRB7=1 ALLARME
(Stesso formato di READ)

PORTA SERIALE (SDR)

Sistema bufferizzato di registri di scorrimento sincrono a 8 bit. Il modo ingresso/uscita e' selezionato da un bit di controllo. Nel modo ingresso, il dato presente sul pin di SP e' trasferito nel registro di scorrimento sul fianco dell'impulso in salita del segnale applicato al pin del CNT. Dopo 8 pulsazioni del CNT, il dato contenuto nel registro di scorrimento viene riversato nel Registro Dati Seriali e viene generata un'interruzione. Nel modo uscita, per il generatore di trasmittanza (velocita' di manipolazione di una linea) si usa il Timer A. Il dato e' trasferito sul pin di SP con un tasso di "underflow" pari a 1/2 quello del Timer A. La trasmittanza massima consentita e' 02/4, ma la trasmittanza massima disponibile e' determinata dal carico della linea e dalla velocita' di risposta al dato in ingresso del ricevitore. La trasmissione inizia seguendo una scrittura sul Registro Dati Seriali (ammesso che il Timer A sia in funzione, ed in modo continuo). Il segnale di clock proveniente dal Timer A appare sul pin del CNT come un'uscita. Il dato contenuto nel Registro Dati Seriali viene caricato nel registro di scorrimento, quindi trasferito al pin di SP, dove si verifica una pulsazione del CNT. Il dato trasferito diviene valido sul fianco dell'impulso in discesa del CNT, rimanendo tale fino al fianco dell'impulso in discesa successivo. Dopo 8 pulsazioni successive del CNT, viene generata un'interruzione per indicare che possono essere inviati altri dati. Se il Registro Dati Seriali e' stato caricato con nuove informazioni prima del verificarsi di questa interruzione, i nuovi dati vengono caricati automaticamente nel registro di scorrimento, e la trasmissione continua. Se il microprocessore e' anticipato di un byte rispetto al registro di scorrimento, la trasmissione e' continua. Se dopo l'ottava pulsazione di CNT non ci sono altri dati da trasmettere, CNT ritorna alto e SP rimane al livello del bit dell'ultimo dato trasmesso. Il dato di SDR e' trasferito fuori da MSB (Most Significant Byte-byte piu'

significativo)per primo; anche un dato seriale in ingresso appare in questo formato.
La capacita' bidirezionale della Porta Seriale e del clock di CNT permettono la connessione di molti dispositivi 6526 ad un comune bus di comunicazione seriale, in cui un 6526 funziona come "master" e rappresenta i dati originali ed il Timer, mentre gli altri circuiti 6526 funzionano come "slaves". Sia l'uscita CNT che quella SP sono canali aperti, in modo da consentire la realizzazione di un tale bus comune. Il protocollo per la selezione master/slave puo' essere trasmesso sul bus seriale, oppure attraverso linee dedicate di "handshacking".

REG NOME

C	SDR	S7	S6	S5	S4	S3	S2	S1	S0

CONTROLLO INTERRUZIONE (ICR)

Sul 6526 ci sono 5 livelli di interruzioni: "underflow" dal Timer A, "underflow" dal Timer B, Allarme del TOD, Porta Seriale piena/vuota e FLAG. Un singolo registro provvede alla mascheratura ed all'informazione dell'interruzione. Il Registro Controllo dell'Interruzione e' formato da un registro MASK a sola scrittura e da un registro DATA a sola lettura. Qualunque interruzione imposta il bit corrispondente del registro DATA. Qualunque interruzione abilitata dal registro MASK imposta il bit IR (MSB) del registro DATA ed abbassa il pin di IRQ. In un sistema multicircuito, si puo' interrogare il bit IR per scoprire quale circuito ha generato una richiesta di interruzione. Il registro di interruzione DATA viene azzerato e la linea di trasmissione IRQ ritorna alta, a seguito di una lettura del registro DATA. Poiche' ogni interruzione imposta un bit di interruzione indipendentemente da MASK, ed ognuno di tali bit puo' essere mascherato in maniera selettiva, per evitare la generazione di un'interruzione del microprocessore, e' possibile miscelare interruzioni interrogative ad interruzioni reali. L'interrogazione del bit IR, tuttavia, provoca l'azzeramento del registro DATA; tocca percio' all'Utente proteggere l'informazione contenuta nel registro DATA da qualunque interruzione interrogativa.
Il registro MASK fornisce un comodo controllo dei bit di una singola maschera. Nella scrittura del registro MASK, se il bit 7 (SET/CLEAR) del dato scritto e' a zero, allora viene azzerato ogni bit di maschera scritto a uno, mentre i bit di maschera scritti a zero vengono lasciati immutati. Se il bit 7 del dato scritto e' a UNO, vengono impostati tutti i bit di maschera scritti a uno, lasciando immutati tutti i bit di maschera scritti a zero. Affinche' un indicatore di interruzione imposti IR e generi una Richiesta di Interruzione, si deve impostare il corrispondente bit di MASK.

READ (INT DATA)

REG NOME

D	ICR	IR	0	0	FLG	SP	ALRM	TB	TA

WRITE (INT MASK)

REG NOME

D	ICR	S/C	X	X	FLG	SP	ALRM	TB	TA

REGISTRI DI CONTROLLO

Nel 6526 ci sono due registri di controllo, CRA e CRB, associati rispettivamente al Timer A ed al Timer B. Il formato di ciascun registro e' il seguente:

*CRA

BIT	NOME	FUNZIONE
0	START	1 = Avvia Timer A, 0 = Arresta Timer A. Automaticamente risettato quando si verifica un "underflow" nel modo monostabile
1	PBON	1 = L'uscita del Timer A appare su PB6, 0 = Operazione normale
2	OUTMODE	1 = Bistabile, 0 = Pulsazione
3	RUNMODE	1 = Monostabile, 0 = Continuo
4	LOAD	1 = Caricamento Forzato (ingresso selettore, non avviene memorizzazione, il bit 4 rilegge sempre uno zero, la scrittura di uno zero non ha effetto)
5	INMODE	1 = Il Timer A conta le transizioni positive di CNT, 0 = Il Timer A conta O2 pulsazioni
6	SPMODE	1 = Uscita Porta Seriale (CNT origina il Timer) 0 = Ingresso Porta Seriale (richiede un Timer esterno)
7	TODIN	1 = Richiesta di clock a 50 Hz sul pin di TOD per tempi di precisione 0 = Richiesta di clock a 60 Hz sul pin di TOD per tempi di precisione

*CRB.

BIT	NOME	FUNZIONE
		(I bit CRB0-CRB4 del Timer B sono identici a CRA0-CRA4, ad eccezione del bit 1, che controlla l'uscita del Timer B su PB7)
5,6	INMODE	Questi bit selezionano per il Timer B uno dei quattro seguenti modi di ingresso:

CRB6	CRB5	
0	0	Timer B conta O2 pulsazioni
0	1	Timer B conta le transizioni positive di CNT
1	0	Timer B conta le pulsazioni di "underflow" del Timer A
1	1	Timer B conta le pulsazioni di "underflow" del Timer A mentre CNT e' alto

BIT	NOME	FUNZIONE
7	ALLARME	1 = Imposta l'ALLARME scrivendo sui registri di TOD 0 = Imposta il clock di TOD scrivendo sui registri di quest'ultimo

REG	NAME	TOD IN	SP MODE	IN MODE	LOAD	RUN MODE	OUT MODE	PB ON	START
E	CRA	0=60Hz 1=50Hz	0=INPUT 1=OUTPUT	0=$\phi 2$ 1=CNT	1=FORCE LOAD (STROBE)	0=CONT 1=O.S.	0=PULSE 1=TOGGLE	0=PB$_6$OFF 1=PB$_6$ ON	0=STOP 1=START

└──────────── TA ────────────┘ (IN MODE … START)

REG	NAME	ALARM	IN MODE		LOAD	RUN MODE	OUT MODE	PB ON	START
F	CRB	0=TOD 1= ALARM	0 1 1 1	0=$\phi 2$ 1=CNT 0=TA 1=CNT TA	1=FORCE LOAD (STROBE)	0=CONT 1=O S	0=PULSE 1=TOGGLE	0=PB$_7$ OFF 1=PB$_7$ ON	0=STOP 1=START

└──────────── TB ────────────┘ (IN MODE … START)

Tutti i bit di un registro che non sono utilizzati vengono lasciati inalterati dalla scrittura e forzati a zero dalla lettura.

Il COMMODORE SEMICONDUCTOR GROUP si riserva il diritto di apportare modifiche ad ogni prodotto illustrato, allo scopo di perfezionarne l'affidabilita', la funzione o il progetto. Il COMMODORE SEMICONDUCTOR GROUP non si assume alcuna responsabilita' derivante dall'applicazione o dall'uso di ogni prodotto o circuito descritto, e non rilascia alcuna licenza sotto i diritti di brevetto propri o altrui.

APPENDICE N

SPECIFICHE DEL CIRCUITO 6566/6567 (VIC-II)

I dispositivi 6566/6567 sono circuiti di controllo del colore del video, di uso generale per quanto riguarda sia le applicazioni di terminali video, sia i Video Games. Entrambi i dispositivi contengono 47 registri di controllo, accessibili da un bus standard a 8 bit (65XX) del microprocessore, che accedono fino a 16K di memoria per le informazioni del video. I vari modi di operare e le opzioni di ogni modo sono descritte di seguito.

MODO CARATTERE VIDEO

Questo modo consente al 6566/6567 di prelevare i PUNTATORI CARATTERE dall'area della MATRICE VIDEO della memoria, convertendoli negli indirizzi locazione carattere a punti nell'area di memoria CARATTERE BASE ampia 2048 bytes. La matrice video e' composta da 1000 locazioni consecutive della memoria, contenenti ognuna un puntatore carattere a 8 bit. La locazione della matrice video all'interno della memoria e' definita nel registro 24 ($18 HEX) da VM13-VM10, utilizzati come i 4 MSB dell'indirizzo della matrice video. I 10 bit bassi sono provvisti di un contatore interno (VC3-VC0), che avanza attraverso le 1000 locazioni carattere. Da notare che il 6566/6567 mette a disposizione 14 uscite indirizzo; per una completa decodifica della memoria del sistema, pertanto, puo'essere necessaria una parte aggiuntiva hardware del sistema.

INDIRIZZI DEL PUNTATORE CARATTERE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Il puntatore carattere a 8 bit consente la disponibilita' simultanea fino a 256 differenti definizioni carattere. Ogni carattere e' una matrice di 8 punti X 8, registrata nella base carattere sotto forma di 8 bytes consecutivi. La locazione della base carattere e' definita nel registro 24 da CB13-CB11, usati per i 3 bit piu' significativi dell'indirizzo della base carattere. Gli 11 indirizzi bassi sono formati dal puntatore carattere a 8 bit proveniente dalla matrice video (D7-D0), che seleziona un particolare carattere, e da un contatore del quadro televisivo a 3 bit, che seleziona uno degli otto bytes carattere. I caratteri risultanti vengono formattati in 25 righe di 40 colonne ciascuna. Oltre al puntatore carattere a 8 bit, ad ogni locazione della matrice video (ciascuna locazione deve essere larga 12 bit) e' associato un SEMYBYTE COLORE, che definisce per ogni carattere uno fra 16 colori.

INDIRIZZI DEI DATI CARATTERE

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	CB12	CB11	D7	D6	D5	D4	D3	D2	D1	D0	RC2	RC1	RC0

MODO CARATTERE STANDARD (MCM = BMM = ECM = 0)

Questo modo consente agli otto bytes sequenziali della base carattere di essere visualizzati direttamente sulle otto righe di ogni regione carattere. Quando il colore selezionato dal semibyte colore (primo piano) viene visualizzato, a causa della presenza di un bit a 1 (vd. Tavola Codici Colore), un bit a 0 provoca la visualizzazione del colore di fondo #0 (dal registro 33 [$21 HEX]).

FUNZIONE	BIT DEL CARATTERE	COLORE VISUALIZZATO
Fondo	0	Colore di Fondo #0 (registro 33 [$21 HEX])
Primo Piano	1	Colore selezionato dai 4 bit colore

Ogni carattere, percio', possiede un unico colore, determinato dal semibyte colore fra i 16 a disposizione; tutti i caratteri hanno un colore do fondo comune.

MODO CARATTERE MULTICOLORE

Questo modo aumenta la flessibilita' del colore, consentendo la visualizzazoone di caratteri fino a 4 colori, anche se di risoluzione ridotta. Il modo multicolore viene scelto impostando a 1 il bit MCM del registro 22 ($16 HEX); cio' provoca una diversa interpretazione dei dati a punti memorizzati nella base carattere. Se l'MSB del semibyte colore e' a 0, il carattere viene visualizzato come descritto nel modo carattere standard, consentendo una miscelazione dei due modi (mettendo pero' a disposizione solo gli 8 colori di ordine basso). Quando invece l'MSB del semibyte colore e' a 1 (se cioe' MCM e' tale che MSB(CM)=1), i bit carattere sono interpretati nel modo multicolore:

FUNZIONE	COPPIA DI BIT DEL CARATTERE	COLORE VISUALIZZATO
Fondo	00	Colore di Fondo #0 (registro 33 [$21 HEX])
Fondo	01	Colore di Fondo #1 (registro 34 [$22 HEX])
Primo Piano	10	Colore di Fondo #2 (registro 35 [$23 HEX])
Primo Piano	11	Colore specificato dai 3 LSB del semibyte colore

Poiche' sono necessari due bit per la specificazione del colore di un punto, il carattere viene ora visualizzato come una matrice 4 X 8, in cui ogni punto ha una misura orizzontale doppia rispetto al modo standard. Da notare tuttavia che, in questo caso, ogni regione carattere puo' contenere 4 differenti colori, 2 di fondo e 2 di primo piano (vd. priorita' di MOB).

MODO COLORE ESTESO (ECM = 1, BMM = MCM = 0)

Questo modo consente la selezione dei singoli colori di fondo per ogni regione carattere, nella normale risoluzione carattere 8 X 8. Questo modo e' selezionato impostando a 1 il bit ECM del registro 17 ($11 HEX). Il dato carattere a punti viene visualizzato come nel modo carattere standard (il colore di primo piano, determinato dal semibyte colore, viene visualizzato a causa della presenza di un bit dato a 1), ma i 2 MSB del puntatore carattere sono usati per scegliere il colore di fondo di ogni regione carattere, come illustrato qui di seguito:

COPPIA MSB DEL PUNT. CARATTERE	COLORE DI FONDO VISUALIZZATO A CAUSA DEL BIT A 0
00	Colore di Fondo #0 (registro 33 [$21])
01	Colore di Fondo #1 (registro 34 [$22])
10	Colore di Fondo #2 (registro 35 [$23])
11	Colore di Fondo #3 (registro 36 [$24])

Poiche' i due MSB dei puntatori carattere sono usati per l'informazione del colore, sono disponibili solamente 64 differenti definizioni di carattere. Il 6566/6567 forza a 0 CB10 e CB9 indipendentemente dai valori originali del puntatore, in modo che siano accessibili solamente la prime 64 posizioni carattere. Con il Modo Colore Esteso, ogni carattere ha uno dei 16 colori di fondo definibili individualmente, ed uno dei 4 colori di primo piano disponibili.

NOTA: I Modi Colore Esteso e Multicolre non possono essere attivati simultaneamente.

MODO BIT MAP

Nel modo Bit Map, il 6566/6567 preleva i dati dalla memoria con una tecnica diversa, in modo da creare una corrispondenza "uno a uno" tra ogni punto visualizzato ed un bit di memoria. Il modo Bit Map consente una risoluzione di schermo di 320 X 200 (H X V) punti video controllati individualmente. Questo modo viene selezionato impostando a 1 il bit BMM del registro 17 ($11 HEX). L'accesso alla MATRICE VIDEO avviene ancora come nel modo carattere, ma i dati della matrice video non sono piu' interpretati come puntatori carattere, bensi' come dati del colore. Il CONTATORE DELLA MATRICE VIDEO viene quindi usato come indirizzo, per prelevare i dati a punti per il video dal byte 8000 della BASE VIDEO. L'indirizzo base del video e' formato nel modo seguente:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
CB13	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

VCx indica le uscite del contatore della matrice video e RCx il contatore a 3 bit della linea di quadro; CB13 proviene dal registro 24 ($18 HEX). Mentre il contatore di quadro si incrementa di uno per ogni linea orizzontale di video (linea di quadro), il contatore della matrice video avanza attraverso le stesse 40 locazioni per otto linee

di quadro, continuando sulle successive 40 locazioni ogni otto linee. In ognuna delle 8 locazioni sequenziali, questo indirizzamento viene formattato sullo schermo video in un blocco di 8 X 8 punti.

MODO BIT MAP STANDARD (BMM = 1, MCM = 0)

Quando si usa il Modo Bit Map Standard, l'informazione del colore deriva solamente dai dati registati nella matrice video (il semibyte colore e' ignorato). Gli otto bit sono divisi in due semibyte, in modo da consentire, in ognuno dei blocchi di 8 X 8 punti, la scelta di due colori indipendenti. Quando un bit della memoria video e' a 0, il colore del punto di output viene impostato dal semibyte meno significativo (piu' basso). Similmente, un bit della memoria video a 1 sceglie il colore di output determinato dall'MSN (Most Significant Nybble-semibyte piu' significativo).

BIT	COLORE DEL VIDEO
0	Semibyte basso del puntatore della matrice video
1	Semibyte alto del puntatore della matrice video

MODO BIT MAP MULTICOLORE (BMM = MCM = 1)

Questo Modo viene selezionato, insieme al bit BMM, impostando a 1 il bit MCM del registro 22 ($16 HEX). Il Modo Multicolore usa le stesse sequenze di accesso alla memoria viste per il Modo Bit Map Standard, ma interpreta il dato punto nel modo seguente:

COPPIA DI BIT	COLORE DEL VIDEO
00	Colore di Fondo #0 (registro 33 [$21 HEX])
01	Semibyte alto del puntatore della matrice video
10	Semibyte basso del puntatore della matrice video
11	Semibyte colore della matrice video

Da notare che il semibyte colore (DB11-DB8) e' usato per il Modo Bit Map Multicolore. Di nuovo, poiche' si usano due bit per la selezione del colore di un punto, la misura orizzontale di tale punto e' raddoppiata, il che comporta una risoluzione di schermo di 160 X 200 punti (H X V). Utilizzando il Modo Bit Map Multicolore, in ogni blocco di 8 X 8 punti si possono visualizzare tre colori scelti indipendentemente, oltre al colore di fondo.

BLOCCHI DI OGGETTI IN MOVIMENTO

Il Blocco Oggetti Mobili (MOB) e' un particolare tipo di carattere che puo' essere visualizzato in qualunque posizione dello schermo senza le limitazioni tipiche dei Modi Carattere e Bit Map. Si possono visualizzare simultaneamente fino a 18 MOB singoli, ognuno definito da 63 bytes di memoria visualizzati come una schiera di 24 X 21 punti. Una serie di caratteristiche particolari rendono i MOB particolarmente adatti per la grafica su video ed applicazioni ai giochi.

BLOCCO VIDEO DEI MOB

BYTE	BYTE	BYTE
00	01	02
03	04	05
..	..	..
..	..	..
..	..	..
57	58	59
60	61	62

ABILITAZIONE

Ogni MOB puo' essere attivato per il video in maniera selettiva, impostando a 1 il corrispondente bit (MnE) del registro 21 ($15 HEX). Se il bit MnE e' a 0, non avviene alcuna operazione riguardante il MOB disabilitato.

POSIZIONE

Ogni MOB viene posizionato per mezzo del proprio registro posizione X e Y (vd. Mappa Registro), con una risoluzione di 512 posizioni orizzontali X 256 verticali. La posizione di un MOB e' determinata dall'angolo in alto a sinistra della schiera. Le posizioni X da 23 a 347 ($17-$157 HEX) e quelle Y da 50 a 249 ($32-$F9 HEX) rientrano nella parte visibile dello schermo. Poiche' non tutte le posizioni visibili di un MOB sono interamente visibili sullo schermo, i MOB possono essere mossi lentamente fuori e dentro lo schermo video.

COLORE

Ogni MOB ha un proprio registro a 4 bit per la determinazione del colore. I due modi colore di un MOB sono:

1) MOB STANDARD (MnMC = 0)

Un bit del MOB a zero consente ad ogni dato di fondo di "trasparire", mentre lo stesso bit a 1 visualizza il colore del MOB determinato dal corrispondente registro colore

2) MOB MULTICOLORE (MnMC = 1)

Ogni MOB puo' essere selezionato individualmente, come MOB multicolore, per mezzo dei bit MnMC del registro 28 ($1C HEX) del MOB multicolore. Quando il bit MnMC e' a 1, il MOB corrispondente viene visualizzato nel modo multicolore; in questo modo, il dato MOB e' interpretato a coppie (come negli altri modi multicolore) nel modo seguente:

COPPIA DI BIT	COLORE VISUALIZZATO
00	Trasparente
01	MOB Multicolore #0 (registro 37 [$25 HEX])
10	MOB Colore (registri 39-46 [$27-$2E HEX])
11	MOB Multicolore #1 (registro 38 [$26 HEX])

Poiche' per ogni colore sono necessari due bit, la risoluzione di un MOB viene ridotta a 12 X 21, dove ogni punto orizzontale e' espanso al

doppio della misura standard, in modo da non modificare la grandezza globale del MOB. Da notare che in ogni MOB si possono visualizzare fino a tre colori (oltre al trasparente), ma due di essi vengono spartiti fra tutti i MOB nel Modo Multicolore.

INGRANDIMENTO

Ogni MOB puo' essere ingrandito singolarmente (2X) in entrambi le direzioni (orizzontale e verticale). Due registri contengono i bit di controllo (MnXE,MnYE) dell'ingrandimento:

REGISTRO	FUNZIONE
23 ($17)	Espansione orizzontale MnxE - 1=Espande, 0=Normale
29 ($1D)	Espansione verticale MnYE - 1=Espande, 0=Normale

Quando un MOB viene ingrandito, non si realizza alcun aumento di risoluzione. Viene visualizzata la stessa schiera 24 X 21 (q2 X 21 se Multicolore)I ma la dimensione globale del MOB viene raddoppiata lungo la direzione desiderata (se un MOB e' contemporaneamente Multicolore ed ingrandito, il punto piu' piccolo di tale MOB puo' essere fino a 4X le dimensioni di un punto standard).

PRIORITÀ

La priorita' di ogni MOB puo' essere controllata individualmente rispetto alle altre informazioni visualozzate provenienti dai Modi Carattere o Bot Map. La priorita' di ogni MOB viene impostata dal corrispondente bit (MnDP) del registro 27 ($1B HEX) nel modo seguente:

BIT	PRIORITÀ RISPETTO A DATI CARATTERE O BIT MAP
0	Vengono visualizzati dati MOB non trasparenti (MOB davanti)
1	Vengono visualizzati dati MOB non trasparenti solo al posto del colore di fondo #0 o della coppia di bit multicolore 01 (MOB dietro)

PRIORITÀ DATI MOB - VIDEO

MnDP=1	MnDP=0
MOBn Primo Piano Fondo	Primo Piano MOBn Fondo

I bit a 0 del dato MOB (00 nel modo multicolore) sono trasparenti, permettendo cosi' ad ogni altra informazione di essere visualizzata.
I MOB hanno una priorita' fissa gli uni rispetto agli altri; in particolare, il MOB 0 ha priorita' massima ed il MOB 7 ha priorita' minima. Quando i dati di uno o piu' MOB (ad eccezione dei dati trasparenti) sono coincidenti, vengono visualizzati quelli del MOB di numero piu' basso. La priorita' fra due MOB viene esaminata prima della risoluzione della priorita' con dati carattere o bit map.

SCOPERTA DEI PUNTI DI CONTATTO

Si possono scoprire due punti di contatto: MOB contro MOB e MOB contro dati sul video.

1) Si ha una collisione fra due MOB quando coincidono dati non trasparenti di output di tali MOB. La coincidenza delle aree trasparenti dei MOB non genera contatto. Quando si genera un contatto, i bit del registro 30 ($1E HEX) CONTATTO MOB-MOB del MOB vengono impostati a 1 per entrambi i MOB interessati dal contatto; lo stesso avviene, per ogni MOB coinvolto, nel caso di contatti fra piu' MOB. I bit di contatto rimangono impostati fino alla prima lettura del registro di contatto, quando tutti i bit sono automaticamente azzerati. I punti di contatto dei MOB vengono scoperti anche per MOB fouri schermo.

2) Il secondo tipo di contatto e' del tipo MOB-DATI; avviene tra un MOB e i dati di primo piano dello schermo provenienti dai modi Carattere o Bit Map. Il registro 31 ($1F HEX) CONTATTO MOB-DATI ha un bit (MnD) per ogni MOB, che viene imposrtato a 1 quando coincidono il MOB e i dati dello schermo non di fondo. Di nuovo, la coincidenza di dati trasparenti non genera contatto. Per applicazioni particolari, i dati del video provenienti dalla coppia di bit multicolore 0-1 non causano un contatto. Cio' consente il loro uso come dati dello schermo di fondo senza che interferiscano nei contatti reali dei MOB. Un contatto MOB-DATI puo' avvenire fuori schermo in direzione orizzontale se il dato attuale dello schermo e' stato fatto scorrere fuori schermo (vd. "scrolling"). Il registro CONTATTO MOB-DATI si azzera automaticamente quando viene letto.

I circuiti "latch" di interruzione del contatto vengono impostati quando il primo bit di entrambi i registri e' impostato a 1. Una volta impostato alto qualunque bit di contatto di un registro, i contatti succesivi non impostano il circuito "latch" di interruzione finche' il relativo registro di contatto non viene azzerato da una lettura.

ACCESSO ALLA MEMORIA MOB

I dati di ogni MOB sono registrati in 63 bytes consecutivi della memoria. Ogni blocco di dati del MOB e' definito da un puntatore MOB, locato alla fine della matrice video. Nei Modi video normali, si usano solamente 1000 bytes della memoria video, in modo da consentire alle locazioni 1016-1023 [da (VMbase+$3F8) a (VMbase+$3FF)] di essere usate per i puntatori MOB 0-7. Il puntatore a 8 bit proveniente dalla matrice video, insieme ai 6 bit del contatore di byte del MOB (usato per indirizzare i 63 bytes), definiscono l'intero campo indirizzo a 14 bit:

A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
MP7	MP6	MP5	MP4	MP3	MP2	MP1	MP0	MC5	MC4	MC3	MC2	MC1	MC0

MPx sono i bit del puntatore MOB proveniente dalla matrice video e MCx sono i bit del contatore MOB generato internamente. I puntatori MOB sono letti dalla matrice video alla fine di ogni linea del quadro. Quando il registro posizione Y coincide con il valore corrente della linea di quadro, iniziano i prelievi effettivi dei dati del MOB. I

contatori interni attraversano automaticamente i 63 byte dei dati del MOB, visualizzando 3 bytes per ogni linea del quadro.

ALTRE CARATTERISTICHE

AZZERAMENTO DEL VIDEO

Il video puo' essere azzerato impostando a zero il bit DEN del registro 17 ($11 HEX). Quando viene azzerato, l'intero schermo viene riempito con il colore esterno, come impostato nel registro 32 ($20 HEX). Diurante l'azzeramento, sono richiesti solamente accessi alla memoria trasparente (Fase 1), in modo da ottener una piena utilizzazione del processore del bus di sistema. Tuttavia, i dati del MOB consentono l'accesso, ammesso che anch'essi non siano stati disabilitati. Per un'immagine video normale, il bit DEN deve essere impostato a 1.

SELEZIONE RIGA/COLONNA

Il video normale e' costituito da 25 righe di 40 caratteri (o regioni carattere ciascuna). Per particolari applicazioni del video, quest'ultimo puo' essere ridotto a 24 righe per 38 caratteri. Il formato dell'informazione visualizzata non subisce alcun cambiamento, eccezion fatta per i caratteri (bit) adiacenti al bordo esterno, che vengono ricoperti dal bordo stesso. I bit di selezione operano nel modo seguente:

RSEL	NUMERO DI RIGHE	CSEL	NUMERO DI COLONNE
0	24	0	38
1	25	1	40

Il bit RSEL risiede nel registro 17 ($11 HEX), ed il bit CSEL nel registro 22 ($16 HEX). Con il video standard si usa di solito la finestra video piu' grande, mentre quella piu' piccola si usa in genere in congiunzione con lo scrolling.

SCROLLING

I dati del video possono essere mossi verso il basso di un intero spazio caratter, sia in orizzontale che in verticale. Quando viene usato in congiunzione con la finestra video piu' piccola, lo "Scrolling puo' essere usato per creare un lento movimento panoramico dei dati sul video, durante l'aggiornamento della memoria del sistema, solamente quando sia richiesta una nuova riga (o colonna) carattere. Lo "scrolling" viene anche usato per centrare uno schermo fisso all'interno della finestra video.

BIT	REGISTRI	FUNZIONE
X2,X1,X0	22 ($16 HEX)	Posizione orizzontale
Y2,Y1,Y0	17 ($11 HEX)	Posizione verticale

PENNA OTTICA

L'ingresso penna ottica registra in un circuito "latch" sul fianco di un impulso in caduta la corrente posizione dello schermo, utilizzando una coppia di registri (LPX, LPY). Il registro 19 ($13 HEX) posizione X contiene gli 8 MSB della posizione X all'istante della tansizione. Poiche' la posizione X e' definita da un contatore a 512 posizioni (9 bit), viene formata una risoluzione di due punti orizzontali. Analogamente, la posizione Y viene registrata nel circuito "latch" del registro 20 ($14 HEX); in questo caso, gli 8 bit forniscono, all'interno dello schermo visibile, una risoluzione di quadro singola. Il circuito "latch" della penna ottica puo' essere "triggerato" solamente una volta per quadro, per cui tutti gli scatti seguenti non avranno alcun effetto. Occorre percio' eseguire diverse prove (mediamente da 3 in su) primA di iniziare ad operare sullo schermo con la penna ottica; il numero delle prove da eseguire varia in base alle caratteristiche della penna ottica impiegata.

REGISTRO DI QUADRO

Il registro di quadro svolge una doppia funzione. La lettura del registro di quadro 18 ($12 HEX) ritorna gli otto bit bassi della corrente posizione del quadro (MSB-RC8 e' locato nel registro 17 [$11 HEX]). Il registro di quadro puo' essere interrogato per l'implementazione di modifiche del video fuori dall'area visibile, per evitare lo sfarfallamento del quadro. La finestra visibile si estende dalla finestra 51 alla finestra 251 ($033-$0F3 HEX). La scrittura dei bit del quadro (comprendente RC8) viene registrata in un circuito "latch", per consentire il loro uso in confronto interno di quadro. Quando il quadro corrente corrisponde al valore registrato, viene impostato il circuito "latch" di interruzione del quadro.

REGISTRO DI INTERRUZIONE

Il Registro di Interruzione mostra lo stato delle quattro sorgenti di interruzione. Le quattro sorgenti di interruzione sono le seguenti:

BIT DI «LATCH»	BIT DI ABILIT.	IMPOSTATO QUANDO:
IRST	ERST	Conteggio del quadro=Conteggio registrato del quadro
IMDC	EMDC	Viene impostato il registro di collisione MOB-DAT((solo per il primo contatto)
IMMC	EMMC	Viene impostato il registro di collisione MOB-MOB (solo per il primo contatto)
ILP	ELP	Si verifica una transizione negativa dell'ingresso LP (una volta per quadro)
IRQ		Viene impostato alto ed abilitato dalla impostazione del circuito "latch" (inverso dell'uscita IRQ/)

Per fare si' che una richiesta di interruzione imposti a zero l'uscita IRQ/, occorre impostare a 1 il corrispondente bit di abilitazione dell'interruzione posto nel registro 26 ($1A HEX). Una volta impostato, il circuito "latch" di interruzione puo' essere azzerato solamente scrivendo a 1 il corrispondente bit di "latch" nel registro interruzione. Questa cartteristica consente la gestione selettiva delle interruzioni video, senza ricorrere al software per

"ricordare" le interruzioni attive.

RICARICA DINAMICA DELLA RAM

I dispositivi 6566/6567 comprendono un sistema di controllo per la ricarica dinamica della RAM. Per ogni linea del quadro vengono ricaricati 5 indirizzi di riga di 8 bit ciascuno. Questo rapporto garantisce un ritardo massimo di 2.02 msec nella ricarica di ogni singolo indirizzo di riga per uno schema di ricarica a 128 indirizzi (per uno schema a 256 indirizzi, il ritardo massimo di ricarica e' di 3.66 msec). La ricarica e' totalmente trasparente al sistema, in quanto avviene durante la Fase 1 del clock di sistema. Il 6567 genera sia RAS/ che CAS/, che di norma sono connessi direttamente alla RAM dinamica. RAS/ e CAS/ vengono generati per ogni Fase 2 ed ogni accesso ai dati video (compresa la ricarica), in modo da non richiedere la generazione di un clock esterno.

RIPRISTINO

Il bit di ripristino (RES) posto nel registrio 32 ($20 HEX) non viene usato per le normali operazioni. Percio' deve essere impostato a zero all'inizializzazione del circuito video. Quando viene impostato a 1, viene sospesa l'intera operazione del circuito video, comprese uscite e sincronizzazione video, ricarica della memoria ed accesso al bus di sistema.

TEORIA DELL'OPERAZIONE

INTERFACCIA DI SISTEMA

I dispositivi di controllo del video del 6566/6567 interagiscono in maniera particolare con il bus dati del sistema. Un sistema 65XX richiede i bus di sistema solo durante la Fase 2 (clock alto) del ciclo. I dispositivi 6566/6567 traggono vantaggio da questa caratteristica per accedere la memoria di sistema durante la Fase 1 (clock basso) del ciclo. Percio', operazioni come il trasporto dei dati o la ricarica della memoria sono totalmente trasparenti al processore, e non ne riducono il rendimento funzionale (trhoughput). I circuiti del video forniscono i segnali di controllo dell'interfaccia necessari al mantenimento di questa condivisione del bus.

I dispositivi del video forniscono il segnale AEC (controllo di abilitazione dell'indirizzo), usato per disabilitare i circuiti pilota del bus indirizzo del processore, i quali consentono al dispositivo video di accedere al bus indirizzo. AEC e' attivo basso, cosi' da permettere la connessione diretta all'ingresso AEC della famiglia 65XX. Il segnale AEC viene normalmente attivato durante la Fase 1, in modo da non influenzare il funzionamento del processore. A causa di questa "condivisione" del bus, tutti gli accessi alla memoria devono essere completati in 1/2 ciclo. Poiche' i circuiti video generano un clock oscillante a 1 MHz (che deve essere usato come Fase 2 del sistema), un ciclo di memoria dura circa 500 msec, compresa la messa a punto dell'indirizzo, l'accesso ai dati e la messa a punto del dispositivo di lettura.

Alcune operazioni del 6566/6567 richiedono dati ad una velocita'

maggiore di quanto disponibile, leggendo solamente durante il tempo della Fase 1; in particolare, cio' vale per l'accesso ai puntatori carattere della matrice video e per il trasporto dei dati MOB. Durante la Fase 2, percio', il processore deve essere disabilitato per consentire l'accesso ai dati. Si ottiene cio' per mezzo del segnale di BA (Bus Available - bus disponibile). In genere la linea BA e' alta, ma durante la Fase 1 viene abbassata per indicare che il circuito video sta per richiedere un accesso ai dati della Fase 2. Per completare tutti gli accessi alla memoria corrente, il processore ha a disposizione tre tempi della Fase 2 dopo che BA e' stato abbassato. Alla quarta Fase 2 dopo che BA e' stato abbassato, il segnale AEC rimane basso durante la Fase 2, quando il circuito preleva i dati. La linea BA e' normalmente connessa all'ingresso RDY di un processore 65XX. I prelievi del puntatore carattere avvengono ogni otto linee di quadro durante l'attivazione di una finestra video, e richiede 40 accessi consecutivi alla Fase 2 per prelevare i puntatori della matrice video. I prelievi dei dati MOB richiedono i quattro seguenti accessi di memoria:

FASE	DATI	CONDIZIONE
1	Puntatore MOB	Tutti i quadri
2	Byte 1 del MOB	Ogni quadro durante la visualizzazione del MOB
1	Byte 2 del MOB	Ogni quadro durante la visualizzazione del MOB
2	Byte 3 del MOB	Ogni quadro durante la visualizzazione del MOB

I puntatori dei MOB vengono prelevati durante ogni altra Fase 1 alla fine di ogni linea di quadro. Come richiesto, i cicli aggiuntivi siono usati per il prelievo dei dati MOB. Nuovamente, tutti i controlli del bus vengono forniti dai dispositivi 6566/6567.

INTERFACCIA DELLA MEMORIA

Le due versioni del circuito interfaccia video, 6566 e 6567, differiscono nelle configurazioni di uscita dell'indirizzo. Il 6566 possiede 13 indirizzi completamente decodificati per la connessione diretta al bus indirizzi del sistema. Il 6567 ha indirizzi multiplexati per la connessione diretta alle RAM dinamiche di 64K. I bit indirizzo meno significativi, A06-A00, sono presenti su A06-A00 mentre e' tenuto basso RAS/, mentre i bit piu' significativi, A13-A08, sono presenti su A05-A00 mentre e' tenuto basso CAS/. I pin A11-A07 sul 6567 sono uscite indirizzo fisse, in modo da permettere la connessione diretta di questi bit ad una ROM convenzionale di 16K (2K X 8) (gli indirizzi bassi necessitano di una registrazione su un circuito "latch" esterno).

INTERFACCIA DEL PROCESSORE

A parte gli accessi speciali alla memoria precedentemente descritti, ai registri del 6566/6567 si puo' accedere in maniera analoga a qualunque altro dispositivo periferico. Vengono forniti i seguenti segnali di interfaccia del processore:

BUS DATI (DB7-DB0)

Gli otto pin del bus dati costituiscono la porta dati bidirezionale, controllata da CS/, RW e dalla Fase 0. Si puo' accedere al bus dati

solo mentre sono alti AEC e Fase 0, e CS/ e' basso.

SELEZIONE CIRCUITO (CS/)

Il pin di selezione circuito, CS/, e' tenuto basso per abilitare l'accesso ai registri del dispositivo, in congiunzione ai pin indirizzo e RW. Il riconoscimento di un CS/ basso avviene solamente quando AEC e Fase 0 sono alti.

LETTURA/SCRITTURA (R/W)

L'ingresso lettura/scrittura R/W e' usato per determinare la direzione del trasferimento del bus dati, in congiunzione con CS/. Quando R/W e' alto (1), i dati sono trasferiti dal registro selezionato all'uscita del bus dati. Quando R/W e' basso (0), i dati presentati sul pin del bus dati sono registrati nel registro selezionato.

BUS INDIRIZZI (A05-A00)

I 6 pin di indirizzo basso, A05-A00, sono bidirezionali. Durante la lettura o la scrittura del dispositivo video da parte del processore, questi pin indirizzo sono visti come uscite. Il dato presente sulle uscite indirizzo sceglie il registro di lettura o scrittura come definito nella mappa dei registri.

USCITA OROLOGIO (PHO)

L'uscita orologio, Fase 0, e' il clock oscillante a 1 MHz usato come ingresso della Fase 0 dal processore 65XX. Tutte le attivita' del bus di sistema sono riferite a questo clock, la cui frequenza viene generata dividendo per 8 il clock a 8 MHz dell'ingresso del video.

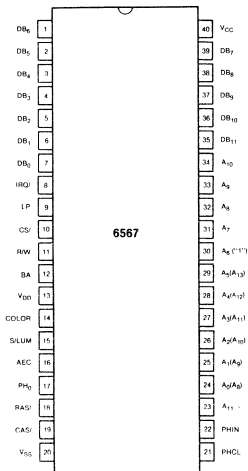
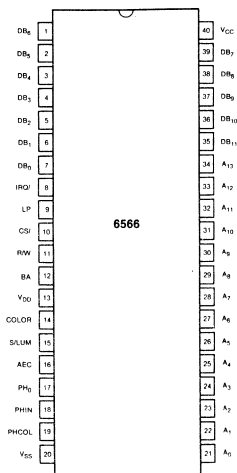
INTERRUZIONI (IRQ)

Quando all'interno del dispositivo si verifica l'abilitazione di una sorgente di interruzione, l'uscita interruzione IRQ/ viene tenuta bassa. Questa uscita e' un canale aperto, che richiede un resistore di pull-up.

INTERFACCIA VIDEO

Il segnale video in uscita dal 6566/6567 e' formato da due segnali che, all'esterno, possono essere miscelati. L'uscita SYNC/LUM contiene tutti i dati del video, comprese la sincronizzazione verticale e la luminosita' dello schermo video. SYNC/LUM e' un canale aperto, che richiede un pull-up da 500 Ohm. L'uscita COLORE contiene tutte le informazioni cromatiche, compresi il contrasto del colore ed il colore di tutti i dati del video. L'uscita COLORE e' una sorgente aperta, che deve essere messa a terra con una resistenza di 1000 Ohm. Il segnale risultante da un'oculata miscelazione di questi segnali puo' pilotare direttamente un monitor per l'uso con un televisore standard.

RIASSUNTO DELL'ATTIVITÀ DEL BUS DEL 6566/6567

AEC	PHO	CS/	R/W	AZIONE
0	0	X	X	Prelievo e ricarica della Fase 1
0	1	X	X	Prelievo della Fase 2 (processore OFF)
1	1	0	0	Scrittura sul registro selezionato
1	1	0	1	Lettura dal registro selezionato
1	1	1	X	Nessuna azione

REGISTER MAP

ADDRESS		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
00	($00)	M0X7	M0X6	M0X5	M0X4	M0X3	M0X2	M0X1	M0X0	MOB 0 X-position
01	($01)	M0Y7	M0Y6	M0Y5	M0Y4	M0Y3	M0Y2	M0Y1	M0Y0	MOB 0 Y-position
02	($02)	M1X7	M1X6	M1X5	M1X4	M1X3	M1X2	M1X1	M1X0	MOB 1 X-position
03	($03)	M1Y7	M1Y6	M1Y5	M1Y4	M1Y3	M1Y2	M1Y1	M1Y0	MOB 1 Y-position
04	($04)	M2X7	M2X6	M2X5	M2X4	M2X3	M2X2	M2X1	M2X0	MOB 2 X-position
05	($05)	M2Y7	M2Y6	M2Y5	M2Y4	M2Y3	M2Y2	M2Y1	M2Y0	MOB 2 Y-position
06	($06)	M3X7	M3X6	M3X5	M3X4	M3X3	M3X2	M3X1	M3X0	MOB 3 X-position
07	($07)	M3Y7	M3Y6	M3Y5	M3Y4	M3Y3	M3Y2	M3Y1	M3Y0	MOB 3 Y-position
08	($08)	M4X7	M4X6	M4X5	M4X4	M4X3	M4X2	M4X1	M4X0	MOB 4 X-position
09	($09)	M4Y7	M4Y6	M4Y5	M4Y4	M4Y3	M4Y2	M4Y1	M4Y0	MOB 4 Y-position
10	($0A)	M5X7	M5X6	M5X5	M5X4	M5X3	M5X2	M5X1	M5X0	MOB 5 X-position
11	($0B)	M5Y7	M5Y6	M5Y5	M5Y4	M5Y3	M5Y2	M5Y1	M5Y0	MOB 5 Y-position
12	($0C)	M6X7	M6X6	M6X5	M6X4	M6X3	M6X2	M6X1	M6X0	MOB 6 X-position
13	($0D)	M6Y7	M6Y6	M6Y5	M6Y4	M6Y3	M6Y2	M6Y1	M6Y0	MOB 6 Y-position
14	($0E)	M7X7	M7X6	M7X5	M7X4	M7X3	M7X2	M7X1	M7X0	MOB 7 X-position
15	($0F)	M7Y7	M7Y6	M7Y5	M7Y4	M7Y3	M7Y2	M7Y1	M6Y0	MOB 7 Y-position
16	($10)	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSB of X-position
17	($11)	RC8	ECM	BMM	DEN	RSEL	Y2	Y1	Y0	See text
18	($12)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raster register
19	($13)	LPX8	LPX7	LPX6	LPX5	LPX4	LPX3	LPX2	LPX1	Light Pen X
20	($14)	LPY7	LPY6	LPY5	LPY4	LPY3	LPY2	LPY1	LPY0	Light Pen Y
21	($15)	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	MOB Enable
22	($16)	—	—	RES	MCM	CSEL	X2	X1	X0	See text
23	($17)	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	MOB Y-expand
24	($18)	VM13	VM12	VM11	VM10	CB13	CB12	CB11	—	Memory Pointers
25	($19)	IRQ	—	—	—	ILP	IMMC	IMBC	IRST	Interrupt Register
26	($1A)	—	—	—	—	ELP	EMMC	EMBC	ERST	Enable Interrupt
27	($1B)	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	MOB-DATA Priority
28	($1C)	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	MOB Multicolor Sel
29	($1D)	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	MOB X-expand
30	($1E)	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	MOB-MOB Collision
31	($1F)	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	MOB-DATA Collision
32	($20)	—	—	—	—	EC3	EC2	EC1	EC0	Exterior Color
33	($21)	—	—	—	—	B0C3	B0C2	B0C1	B0C0	Bkgd #0 Color
34	($22)	—	—	—	—	B1C3	B1C2	B1C1	B1C0	Bkgd #1 Color
35	($23)	—	—	—	—	B2C3	B2C2	B2C1	B2C0	Bkgd #2 Color
36	($24)	—	—	—	—	B3C3	B3C2	B3C1	B3C0	Bkgd #3 Color
37	($25)	—	—	—	—	MM03	MM02	MM01	MM00	MOB Multicolor #0
38	($26)	—	—	—	—	MM13	MM12	MM11	MM10	MOB Multicolor #1
39	($27)	—	—	—	—	M0C3	M0C2	M0C1	M0C0	MOB 0 Color
40	($28)	—	—	—	—	M1C3	M1C2	M1C1	M1C0	MOB 1 Color
41	($29)	—	—	—	—	M2C3	M2C2	M2C1	M2C0	MOB 2 Color
42	($2A)	—	—	—	—	M3C3	M3C2	M3C1	M3C0	MOB 3 Color
43	($2B)	—	—	—	—	M4C3	M4C2	M4C1	M4C0	MOB 4 Color
44	($2C)	—	—	—	—	M5C3	M5C2	M5C1	M5C0	MOB 5 Color
45	($2D)	—	—	—	—	M6C3	M6C2	M6C1	M6C0	MOB 6 Color
46	($2E)	—	—	—	—	M7C3	M7C2	M7C1	M7C0	MOB 7 Color

NOTA: Il trattino indica l'assenza di connessioni. Tutte le assenze di connessioni sono lette come 1.

CODICI DEL COLORE

D4	D3	D2	D1	D0	HEX	DEC	COLORE
0	0	0	0	0	0	0	Nero
0	0	0	0	1	1	1	Bianco
0	0	0	1	0	2	2	Rosso
0	0	0	1	1	3	3	Azzurro
0	0	1	0	0	4	4	Porpora
0	0	1	0	1	5	5	Verde
0	0	1	1	0	6	6	Blu
0	0	1	1	1	7	7	Giallo
0	1	0	0	0	8	8	Arancio
0	1	0	0	1	9	9	Marrone
0	1	0	1	0	A	10	Rosso chiaro
0	1	0	1	1	B	11	Grigio scuro
0	1	1	0	0	C	12	Grigio medio
0	1	1	0	1	D	13	Verde chiaro
0	1	1	1	0	E	14	Blu chiaro
0	1	1	1	1	F	15	Grigio chiaro

APPENDICE O

SPECIFICHE CIRCUITO 6581 DISPOSITIVO INTERFACCIA DEL SUONO

CONCETTO

Il dispositivo 6581 Interfaccia del Suono (SID) e' un generatore monochip di effetti suono/sintetizzatore di musica elettronica a tre voci, compatibile con le famiglie di processori 65XX e simili. Il SID fornisce un controllo di nota (frequenza) ampio e di alta risoluzione, un colore di tono (indice armonico) ed una dinamica (volume). Un complesso di circuiti di controllo specializzati minimizza il carico di software, semplificandone l'uso nei video games ed in strumenti musicali di costo contenuto.

CARATTERISTICHE

* Oscillatori a 3 toni
  Frequenza 0..4 KHz
* 4 forme d'onda per oscillatore
  Triangolare, "dente di sega", pulsazione variabile, rumore
* 3 modulatori d'ampiezza
  Definizione: 48 db
* 3 generatori di inviluppo
  Risposta esponenziale
  Intervallo di ATTACCARE: 2 msec - 8 sec
  Intervallo di DECADERE: 6 msec - 24 sec
  Livello di SOSTENERE: 0 - volume di picco
  Intervallo di RILASCIARE: 6 msec - 24 sec
* Sincronizzazione dell'oscillatore
* Modulazione circolare
* Filtro programmabile
  Frequenza di taglio: 30 Hz - 12 KHz
  Rolloff a 12 db/ottava
  Uscite passa alto, passa basso, passa banda, taglio
  Risonanza variabile
* Controllo del volume del master
* 2 interfacce A/D POT
* Generatore di numeri/modulazioni casuali
* Ingresso audio esterno

PIN CONFIGURATION

6581 BLOCK DIAGRAM

DESCRIZIONE

Il 6581 e' formato da tre "voci" del sintetizzatore, che possono essere usate indipendemente o congiuntamente le une con le altre (o con sorgenti audio esterne) per la creazione di suoni complessi. Ogni voce e' formata da un Generatore Forma d'Onda/Oscillatore di Tono, un Generatore di Inviluppo ed un modulatore d'ampiezza. L'Oscillatore di Tono controlla la nota della voce in un ampio intervallo; produce anche quattro forme d'onda alla frequenza scelta, con la capacita' armonica unica di fornire ad ogni forma d'onda un semplice controllo del colore di tono. La dinamica del volume dell'oscillatore e' controllata, sotto la direzione del Generatore di Inviluppo, dal Modulatore di Ampiezza. Quando viene attivato, il Generatore di inviluppo crea un inviluppo di ampiezze di volume programmabile ascendente e discendente. Oltre alle tre voci, viene messo a disposizione un Filtro programmabile per la generazione di colori di tono complessi e dinamici, ottenibili attraverso la sintesi sottrattiva.
Il SID permette al microprocessore di leggere le uscite in via di cambiamento del terzo Oscillatore e del terzo Generatore di inviluppo. Tali uscite possono essere usate come sorgente di informazioni di modulazione per creare effetti di vibrato, rapidi movimenti frequenza/filtro e simili. Il terzo oscillatore puo' anche funzionare come generatore di numeri casuali per i giochi. Due convertitori A/D consentono l'interfacciamento del SID con dei potenziometri. Questi convertitori possono essere usati in un gioco per i "paddle", o come controlli del pannello frontale in un sinetizzatore musicale. Il SID puo' elaborare segnali audio esterni, consentendo a piu' circuiti SID di essere collegati a "daisy-chain" o miscelati in complessi sistemi polifonici.

REGISTRI DI CONTROLLO DEL SID

La generazione del suono e' controllata nel SID da 29 registri a 8 bit; questi registri, elencati sotto, sono sia a Sola Lettura che a sola Scrittura.

	ADDRESS					REG #	DATA								REG NAME	REG
	A_4	A_3	A_2	A_1	A_0	(HEX)	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Voice 1**	TYPE
0	0	0	0	0	0	00	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
1	0	0	0	0	1	01	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
2	0	0	0	1	0	02	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
3	0	0	0	1	1	03	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
4	0	0	1	0	0	04	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
5	0	0	1	0	1	05	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
6	0	0	1	1	0	06	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Voice 2	
7	0	0	1	1	1	07	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
8	0	1	0	0	0	08	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
9	0	1	0	0	1	09	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
10	0	1	0	1	0	0A	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
11	0	1	0	1	1	0B	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
12	0	1	1	0	0	0C	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
13	0	1	1	0	1	0D	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Voice 3	
14	0	1	1	1	0	0E	F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0	FREQ LO	WRITE ONLY
15	0	1	1	1	1	0F	F_{15}	F_{14}	F_{13}	F_{12}	F_{11}	F_{10}	F_9	F_8	FREQ HI	WRITE ONLY
16	1	0	0	0	0	10	PW_7	PW_6	PW_5	PW_4	PW_3	PW_2	PW_1	PW_0	PW LO	WRITE ONLY
17	1	0	0	0	1	11	—	—	—	—	PW_{11}	PW_{10}	PW_9	PW_8	PW HI	WRITE ONLY
18	1	0	0	1	0	12	NOISE	⊓⊓	/|/|	/\/\	TEST	RING MOD	SYNC	GATE	CONTROL REG	WRITE ONLY
19	1	0	0	1	1	13	ATK_3	ATK_2	ATK_1	ATK_0	DCY_3	DCY_2	DCY_1	DCY_0	ATTACK/DECAY	WRITE ONLY
20	1	0	1	0	0	14	STN_3	STN_2	STN_1	STN_0	RLS_3	RLS_2	RLS_1	RLS_0	SUSTAIN/RELEASE	WRITE ONLY
															Filter	
21	1	0	1	0	1	15	—	—	—	—	—	FC_2	FC_1	FC_0	FC LO	WRITE ONLY
22	1	0	1	1	0	16	FC_{10}	FC_9	FC_8	FC_7	FC_6	FC_5	FC_4	FC_3	FC HI	WRITE ONLY
23	1	0	1	1	1	17	RES_3	RES_2	RES_1	RES_0	FILTEX	FILT 3	FILT 2	FILT 1	RES/FILT	WRITE ONLY
24	1	1	0	0	0	18	3 OFF	HP	BP	LP	VOL_3	VOL_2	VOL_1	VOL_0	MODE/VOL	WRITE ONLY
															Misc	
25	1	1	0	0	1	19	PX_7	PX_6	PX_5	PX_4	PX_3	PX_2	PX_1	PX_0	POT X	READ ONLY
26	1	1	0	1	0	1A	PY_7	PY_6	PY_5	PY_4	PY_3	PY_2	PY_1	PY_0	POT Y	READ ONLY
27	1	1	0	1	1	1B	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0	OSC_3/RANDOM	READ ONLY
28	1	1	1	0	0	1C	E_7	E_6	E_5	E_4	E_3	E_2	E_1	E_0	ENV_3	READ ONLY

TABELLA 1 - MAPPA DEI REGISTRI DEL SID

DESCRIZIONE DEI REGISTRI DEL SID

VOCE 1

ALTA/BASSA FREQUENZA (REGISTRI 00,01)

L'unione di questi due registri forma un numero a 16 bit che controlla linearmente la frequenza dell'Oscillatore 1, determinata dalla seguente equazione:

Fout=(Fn X Fclk / 16777216) Hz

essendo Fn il numero a 16 bit nei Registri di Frequenza, e Fclk il clock di sistema applicato all'ingresso O2 (pin 6). Per un clock standard a 1 MHz, la frequenza e' data dall'equazione:

Fout=(Fn X 0 059604645) Hz

L'Appendice E fornisce una tavola completa dei valori per la generazione di 8 ottave della scala musicale accordata in maniera uniforme al concerto A (440 Hz). Da notare che la risoluzione della frequenza del SID e' sufficiente per ogni scala di accordi e consente di muoversi di nota in nota (portamento) senza alcun intervallo di frequenza percebile.

AMPIEZZA DI PULSAZIONE ALTA/BASSA (PW LO/PW HI) (REGISTRI 02,03)

L'unione di questi registri forma un numero di 12 bit (i bit 4-7 di PW HI non sono usati) che controlla linearmente l'ampiezza dell'impulso (ciclo di servizio) della forma d'onda dello stesso impulso sull'oscillatore 1. L'ampiezza dell'impulso e' data dall'equazione.

PWout=(PWn/40.95)%

essendo PWn il numero a 12 bit dei registri di ampiezza dell'impulso. La risoluzione dell'ampiezza dell'impulso consente all'ampiezza stessa di essere mossa lentamente senza alcun intervallo percepibile. Un valore 0 o 4095 ($FFF HEX) nei registri di ampiezza dell'impulso producono un'uscita costante DC, mentre un valore 2048 ($800 HEX) produce un'onda quadra.

REGISTRO DI CONTROLLO (REGISTRO 04)

Contiene otto bit di controllo che selezionano varie opzioni sull'oscillatore 1.

GATE (BIT 0) Controlla il Generatore di Inviluppo per la Voce 1. Quando questo bit e' impostato a 1, il Generatore di Inviluppo viene attivato, iniziando il ciclo ATTACCARE/DECADERE/SOSTENERE. Quando il bit viene ripristinato a 0, inizia il ciclo RILASCIARE. Il Generatore di Inviluppo controlla l'ampiezza dell'oscillatore 1 che appare all'uscita audio; percio' il bit GATE deve essere impostato (con gli

adatti parametri di inviluppo), se si vuole che l'uscita scelta dell'oscillatore 1 sia udibile. **Una descrizione** dettagliata del Generatore di Inviluppo si trova alla fine di questa Appendice.

SYNC (BIT 1) Impostato a 1, questo bit sincronizza la frequenza fondamentale dell'oscillatore 1 con quella dell'oscillatore 3, producendo effetti di "Hard Sync".
Variando la frequenza dell'oscillatore 1 rispetto all'oscillatore 3, si produce un vasto rango di complesse strutture armoniche, che va dalla Voce 1 alla frequenza dell'oscillatore 3. Per realizzare la sincronizzazione, l'oscillatore 3 deve essere impostato ad una frequenza diversa da zero, ma preferibilmente piu' bassa di quella dell'oscillatore 1. Nessun altro parametro della Voce 3 ha effetto sulla sincronizzazione.

RING MOD (BIT 2) Quando questo bit e' impostato a 1, la forma d'onda triangolare in uscita dall'oscillatore 1 viene sostituita con una modulazione circolare combinata degli oscillatori 1 e 3. Variando la frequenza dell'oscillatore 1 rispetto all'oscillatore 3, si ottiene un ampio intervallo di strutture non armoniche di ipertoni, per la creazione di suoni di campane o di gong e per effetti speciali. Per rendere udibile la modulazione circolare, occorre scegliere la forma d'onda triangolare dell'oscillatore 1, ed impostare l'oscillatore 3 ad una frequenza diversa da zero. Nessun altro parametro della Voce 3 ha effetto sulla modulazione circolare.

TEST (BIT 3) Se impostato a 1, riposiziona a zero l'oscillatore 1 bloccandolo in questa posizione finche' il bit TEST non viene azzerato. Anche la forma d'onda Rumore in uscita dall'oscillatore 1 viene riposizionata, e la forma d'onda Pulsazione viene tenuta al livello DC. Normalmente, questo bit e' usato per eseguire dei test, tuttavia puo' essere usato per la sincronizzazione dell'oscillatore 1 con eventi esterni, consentendo cosi' di generare forme d'onda altamente complesse sotto il controllo in tempo reale del software.

BIT 4 Se impostato a 1, viene selezionata, in uscita dall'oscillatore 1, la forma d'onda triangolare. Questa forma d'onda ha un'armonica bassa ed una qualita' di suono dolce, simile ad un flauto.

BIT 5 Se impostato a 1, viene selezionata, in uscita dall'oscillatore 1, la forma d'onda a "dente di sega". Questa forma d'onda abbonda di armoniche pari e dispari, presentando un suono brillante come quello degli ottoni.

BIT 6 Se impostato a 1, viene selezionata, in uscita dall'oscillatore 1, la forma d'onda Pulsazione. Il contenuto armonico di questa forma d'onda puo' essere regolato con i registri di Ampiezza della Pulsazione, producendo qualita' di tono che vanno da un'onda quadra cava e brillante ad una pulsazione nasale ed acuta. Muovendo in tempo reale l'ampiezza della pulsazione, si produce un effetto dinamico di "fasatura" che aggiunge al suono un senso di movimento. Saltando rapidamente tra ampiezze di pulsazione, si possono riprodurre interessanti sequenze armoniche.

RUMORE (BIT 7) Quando questo bit e' impostato a 1, viene scelta, in uscita dall'oscillatore 1, la forma d'onda Rumore. Questo e' un segnale casuale che cambia secondo la frequenza dell'oscillatore 1. La

qualita' del suono puo' essere variata, attraverso i registri della Frequenza dell'oscillatore 1. La qualita' del suono puo' essere variata, attraverso i registri della frequenza dell'oscillatore 1, da un rumore sordo e cupo ad un rumore bianco e sibilante. Il Rumore puo' essere usato per creare esplosioni, spari, rumori di motori a reazione, vento, risacca ed altri suoni atonali come batterie e piatti. Muovendo la frequenza dell'oscillatore con il rumore scelto, si produce un effetto di movimento impetuoso.
Una delle forme d'onda dell'oscillatore 1, per essere udibile, deve essere selezionata, tuttavia non e' necessario rimuovere le forme d'onda per togliere il volume all'uscita della Voce 1. L'ampiezza della Voce 1 all'uscita finale e' funzione del solo Generatore di Inviluppo.

NOTA: Le forme d'onda di uscita dell'oscillatore NON sono aggiuntive. Se si sceglie simultaneamente piu' di una forma d'onda in uscita, il risultato e' una AND logica delle forme d'onda. Questa tecnica puo' essere usata per generare forme d'onda addizionali alle quattro sopra descritte, tuttavia e' necessaria una certa attenzione. Se viene scelta qualunque altra forma d'onda mentre e' in funzione il rumore, l'uscita del rumore puo' "chiudersi". In questo caso, quest'uscita rimane silenziosa finche' non viene impostata daccapo dal bit TEST, oppure portando RES (pin 5) basso

ATTACCARE/DECADERE (REGISTRO 05)

I bit 4-7 (ATK0-ATK3) di questo registro scelgono per il Generatore di Inviluppo della Voce 1 una delle 16 classi di ATTACCARE. Tale classe di ATTACCARE determina la rapidita' di crescita dell'uscita della Voce 1 da zero all'ampiezza di picco, quando venga attivato il Generatore di Inviluppo. La Tavola 2 illustra le 16 classi di ATTACCARE
I bit 0-3 (DCY0-DCY3) selezionano per il Generatore di Inviluppo una delle 16 classi di DECADERE. Il ciclo di DECADERE segue quello di ATTACCARE; la classe di DECADERE determina la rapidita' di caduta dell'uscita dall'ampiezza di picco al livello di SOSTENERE scelto. La Tavola 2 illustra le 16 classi di DECADERE.

SOSTENERE/RILASCIARE (REGISTRO 06)

I bit 4-7 (STN0-STN3) di questo registro scelgono per il Generatore di Inviluppo uno dei 16 livelli di SOSTENERE. Il ciclo di SOSTENERE segue quello di DECADERE; l'uscita della Voce 1 rimane all'ampiezza di SOSTENERE scelta per tutto il tempo in cui il bit GATE rimane impostato. I livelli di SOSTENERE coprono un intervallo che va da zero all'ampiezza di picco in 16 passi lineari, dove un valore di SOSTENERE 0 seleziona un'ampiezza 0 ed un valore di SOSTENERE 15 ($F HEX) seleziona l'ampiezza di picco. Il valore di SOSTENERE 8 imposta SOSTENERE della Voce 1 a meta' ampiezza di picco raggiungibile dal ciclo di ATTACCARE.
I bit 0-3 (RLS0-RLS3) scelgono per il Generatore di Inviluppo una delle 16 classi di RILASCIARE. Il ciclo di RILASCIARE segue quello di SOSTENERE, quando il bit GATE e' impostato di nuovo a zero. A questo punto, l'uscita della Voce 1 scende dall'ampiezza di sostegno a zero con la classe di RILASCIARE selezionato. Le 16 classi di RILASCIARE sono identiche alle classi di DECADERE.

NOTA: Il movimento in ciclo del Generatore di Inviluppo puo' essere modificato ad ogni istante attraverso il bit GATE. Ad esempio, se il bit GATE viene modificato prima che l'inviluppo abbia terminato il ciclo di ATTACCARE, il ciclo di RILASCIARE inizia immediatamente, partendo da qualunque ampiezza sia stata raggiunta. Se poi l'inviluppo viene nuovamente attivato (prima che il ciclo di RILASCIARE abbia raggiunto ampiezza zero), inizia un nuovo ciclo di ATTACCARE, partendo da qualunque ampiezza sia stata raggiunta. Questa tecnica puo' essere usata per la generazione di complessi inviluppi di ampiezze attraverso il controllo in tempo reale del software.

QUANTITÀ		ATTACCARE	DECADERE RILASCIARE
DEC	HEX	Tempo/ciclo	Tempo/ciclo
0	0	2 msec	6 ms
1	1	8 msec	24 ms
2	2	16 msec	48 ms
3	3	24 msec	72 ms
4	4	38 msec	114 ms
5	5	56 msec	168 ms
6	6	68 msec	204 ms
7	7	80 msec	240 ms
8	8	100 msec	300 ms
9	9	250 msec	750 ms
10	A	500 msec	1.5 s
11	B	800 msec	2.4 s
12	C	1 sec	3 s
13	D	3 sec	9 s
14	E	5 sec	15 s
15	F	8 sec	24 s

TAVOLA 2 – VALORI DELL'INVILUPPO

NOTA: Le classi di inviluppo sono basate su un clock O2 oscillante ad 1 MHz. Per le altre frequenze di O2, moltiplicare la classe data per 1 MHz/O2. Le classi fanno riferimento all'ammontare del tempo per ciclo. Ad esempio, dato un valore di ATTACCARE uguale a 2, il ciclo di ATTACCARE impiega 16 msec per crescere da zero all'ampiezza di picco. Le classi DECADERE/RILASCIARE si riferiscono alla quantita' di tempo impiegato da questi cicli per decrescere dall'ampiezza di picco a zero.

VOCE 2

I registri $07-$0D controllano la Voce 2; sono funzionalmente identici ai registri $00-$06, da cui differiscono per i seguenti punti:

1) Quando viene scelto, SYNC sincronizza l'oscillatore 2 con l'oscillatore 1.
2) Quando viene scelto, RING MOD sostituisce l'uscita triangolo dell'oscillatore 2 con la modulazione circolare combinata degli oscillatori 2 e 1.

VOCE 3

I registri $0E-$14 controllano la Voce 3; sono funzionalmente identici ai registri $00-$06, da cui differiscono per i seguenti punti:

1) Quando viene scelto, SYNC sincronizza l'oscillatore 3 con l'oscillatore 2
2) Quando viene scelto, RING MOD sostituisce l'uscita triangolo dell'oscillatore 2 con la modulazione circolare combinata degli oscillatori 3 e 2.

L'operazione tipica di una voce consiste nella selezione dei parametri desiderati, cioe' frequenza, forma d'onda, effetti (SYNC, RING MOD) e classi di inviluppo, e nell'attivazione della voce stessa tutte le volte che si desideri quel suono. Quest'ultimo puo' essere sostenuto per qualunque periodo di tempo, e terminato azzerando il bit GATE. Ogni voce puo' essere usata separatamente, con parametri ed attivazioni indipendenti, oppure all'unisono, per la creazione di una singola, potente voce. In quest'ultimo caso, un leggero abbassamento della sintonia di ogni oscillatore, oppure un accordo ad intervalli di musica, crea un suono intenso ed animato.

FILTRO

FREQUENZA DI TAGLIO ALTA/BASSA (FC LO/FC HI) (REGISTRI $15, $16)

Questa coppia di registri forma un numero a 11 bit (i bit 3-7 di FC LO non sono usati), che controlla linearmente la Frequenza di Taglio (o di Centro) del Filtro programmabile. La Frequenza di Taglio copre un intervallo circa da 30 Hz a 12 KHz.

RES/FILT (REGISTRO $17)

I bit 4-7 (RES0-RES3) di questo registro controllano la risonanza del filtro, dove per risonanza si intende un effetto di picco enfatizzante i componenti della frequenza alla Frequenza di Taglio del Filtro, causando un suono piu' acuto. Ci sono 16 impostazioni della risonanza, disposte linearmente da risonanza 0 a risonanza 15 ($F). I bit 0-3 determinano quali segnali vengono instradati attraverso il Filtro:

FILT 1 (BIT 0) Se impostato a 0, la Voce 1 appare direttamente all'uscita audio, senza essere influenzata dal filtro.Se impostato a 1 la Voce 1 viene filtrata ed il suo contenuto armonico alterato secondo i parametri del Filtro scelto.

FILT 2 (BIT 1) Come FILT 1, ma per la Voce 2.

FILT 3 (BIT 2) Come FILT 1, ma per la Voce 3.
FILTEX (BIT 3) Come FILT 1, ma per ingresso audio esterno (pin 26).

MODO/VOLUME (REGISTRO $18)

I bit 4-7 di questo registro scelgono varie opzioni di modo del Filtro e di Uscita:

LP (BIT 4) Se impostato a 1, viene scelta l'uscita passa-basso del Filtro ed inviata all'uscita audio. Per un dato segnale in ingresso

del Filtro, tutte le componenti di una frequenza al di sotto della frequenza di taglio del filtro sono lasciate passare inalterate, mentre tutte quelle al di sopra del Taglio vengono attenuate di 12 db/ottava. Il modo passa-basso produce suoni corposi.

BP (BIT 5) Come LP, ma per l'uscita passabanda. Tutti i componenti di una frequenza sopra e sotto il taglio sono attenuati di 6 db/ottava. Il modo passabanda produce suoni sottili ed aperti.

HP (BIT 6) Come LP, ma per l'uscita passa-alto. Tutti i componenti di una frequenza al di sopra del Taglio sono lasciati passare inalterati, mentre tutti quelli al di sotto vengono attenuati di 12 db/ottava. Il modo passa-alto produce suoni metallici, simili ad un ronzio.

3 OFF (BIT 7) Se impostata a 1, l'uscita della Voce 3 viene staccata dal percorso diretto dell'audio. Impostando la Voce 3 in modo da bypassare il Filtro (FILT 3=0), e 3 OFF a 1, si impedisce alla Voce 3 di raggiungere l'uscita audio. Cio' consente alla Voce 3 di essere usata per scopi di modulazione senza uscite non desiderate.

NOTA: I modi uscita del Filtro SONO additivi, in modo da consentire la selezione simultanea di piu' modi del Filtro. Ad esempio, i modi LP e HP possono essere selezionati in modo da produrre una risposta del Filtro di Taglio (o Rigetto di Banda). Affinche' il Filtro abbia un effetto udibile, deve essere selezionata come minimo un'uscita del Filtro attraverso il quale deve essere inviata almeno una Voce. Percio' il Filtro e' l'elemento piu' importante del SID, in quanto permette di generare colori di tono complesso attraverso la sintesi sottrattiva (il Filtro e' usato per eliminare specifiche componenti di una frequenza da un segnale di ingresso armonicamente ricco). I migliori risultati si ottengono variando in tempo reale la Frequenza di Taglio.

BIT 0–3 (VOL 0 – VOL 3) Selezionano 1 dei 16 livelli complessivi di volume per l'uscita composta audio finale. I livelli di volume dell'uscita vanno da zero al massimo ($F=15) in 16 passi lineari. Questo controllo puo' essere usato come un controllo statico di volume per il bilanciamento dei livelli in sistemi multicircuito, oppure per creare effetti dinamici di volume, come il "Tremolo". Per far si' che il SID produca dei suoni, occorre scegliere dei livelli di volume diversi da zero.

IN GENERALE.....

POTX (REGISTRO $19)

Questo registro consente al microprocessore di leggere la posizione del potenziometro allacciato a POTX (pin 24), con valori che vanno da zero alla minima resistenza e da 255 ($FF HEX) alla massima resistenza. Il valore e' sempre valido ed e' aggiornato ogni 512 cicli del clock O2. Per ulteriori informazioni sui valori di POT e del condensatore, si veda la Descrizione dei Pin.

POTY (REGISTRO $1A)

Come POTX, per il POT allacciato a POT Y (pin 23)

OSC 3/RANDOM (REGISTRO $1B)

Permette al microprocessore di leggere gli 8 bit di uscita piu' alti dell'oscillatore 3. Il carattere dei numeri generati e' messo direttamente in relazione alla forma d'onda scelta. Se si sceglie la forma d'onda a "dente di sega" dell'oscillatore 3, il registro presenta una serie di numeri crescenti da 0 a 255 ($FF HEX) con un incremento determinato dalla frequenza dell'oscillatore 3. Se si sceglie la forma d'onda triangolare, l'uscita cresce da zero a 255 per poi tornare a zero. Se si sceglie la forma d'onda Pulsazione, l'uscita si muove a salti da 0 a 255. Se infine la forma d'onda scelta e' il Rumore, viene generata una sequenza di numeri casuali; percio', questo registro puo' essere usato nella scrittura di giochi come generatore di numeri casuali. Ci sono numerose applicazioni dell'oscillatore 3 per la sincronizzazione e la messa in sequenza, tuttavia la funzione principale e' essenzialmente quella di generatore di modulazione. I numeri generati da questo registro possono essere sommati in tempo reale via software ai registri dell'oscillatore, ai registri di Frequenza del Filtro oppure ai registri di ampiezza della Pulsazione. Molti effetti dinamici possono essere generati in questo modo. Si puo' creare un suono di sirena aggiungendo l'uscita a "dente di sega" dell'oscillatore 3 al controllo di frequenza di un altro oscillatore. Effetti "Sample and Hold" possono essere prodotti aggiungendo l'uscita rumore dell'oscillatore 3 ai registri di controllo di Frequenza del Filtro. Si puo' ottenere il vibrato impostando l'oscillatore 3 ad una frequenza di circa 7 Hz, ed aggiungendo l'uscita Triangolo (con un rapporto di scala adeguato) al controllo di frequenza di un altro oscillatore. Alterando la frequenza dell'oscillatore 3 e regolando l'uscita OSC 3, si puo' generare un numero illimitato di effetti. Normalmente, quando l'oscillatore 3 e' usato per la modulazione, si esclude l'uscita audio della Voce 3 (3 OFF=1)

ENV 3 (REGISTRO $1C)

Analogo a OSC 3, questo registro permette al microprocessore di leggere l'uscita del Generatore di Inviluppo della Voce 3. Quest'ultima puo' essere aggiunta alla Frequenza del Filtro per produrre inviluppi armonici, "wah-wah" ed altri effetti simili. Si possono creare suoni "Phaser" aggiungendo quest'uscita ai registri di controllo della frequenza di un oscillatore. Per produrre un'uscita da questo registro, deve essere attivato il Generatore di Inviluppo della Voce 3. Tuttavia, il registro OSC 3 riflette sempre l'uscita variabile dell'oscillatore; questo registro non viene influenzato in alcun modo dal Generatore di Inviluppo.

DESCRIZIONE DEI PIN DEL SID

CAP1A, CAP1B (PIN 1,2) / CAP2A, CAP2B (PIN 3,4)

Questi pin sono usati per la connessione di condensatori integratori richiesti dal Filtro Programmabile. C1 connette i pin 1 e 2, C2 connette i pin 3 e 4. Entrambi i condensatori devono avere lo stesso valore. Il funzionamento normale del filtro intervallo audio (circa 30...12000 Hz) si ottiene per mezzo di un valore di 2200 pF per C1 e C2. Si consigliano condensatori in polistirolo, ed in complessi

sistemi polifonici, dove piu' circuiti SID si controllano gli uni con gli altri, condensatori accoppiati.
L'intervallo della frequenza del Filtro puo' essere tagliato su misura per applicazioni specifiche scegliendo gli appropriati valori del condensatore. Ad esempio, un gioco di caratteristiche economiche puo' non richiedere una piena risposta sulle alte frequenze. In questo caso, per avere un maggior controllo sulle basse frequenze del Filtro, si possono scegliere valori piu' grandi per C1 e C2. La Frequenza di Taglio massima del Filtro e' data da:

FCmax=2.6 E-5/C

dove C e' il valore del condensatore. L'intervallo del Filtro si estende per 9 ottave al di sotto della Frequenza di Taglio massima.

RES (PIN 5)

Questo ingresso del livello TTL e' il controllo di ripristino per il SID. Quando viene portato basso per un minimo di 10 cicli di O2, tutti i registri interni vengono azzerati e l'uscita audio viene silenziata. Questo pin e' in genere connesso alla linea di ripristino del microprocessore, oppure ad un circuito.

02 (PIN 6)

Questo ingresso del livello TTL costituisce il clock master per il SID. Tutte le freguenze dell'oscillatore e le quantita' dell'inviluppo sono riferite a questo clock. O2 controlla anche i trasferimenti dei dati tra il SID ed il microprocessore; tali dati possono essere trasferiti solo quando O2 e' alto. Essenzialmente, O2 si comporta come un selettore di circuito attivo impostato alto finche' si e' interessati ad un trasferimento di dati. Questo pin e' generalmentne connesso al clock di sistema, con una frequenza nominale di funzionamento di 1 MHz.

R/W (PIN 7)

Questo ingresso del livello TTL controlla la direzione dei trasferimenti dei dati tra il SID ed il microprocessore. Se si sono incontrate le condizioni di scelta del circuito, un alto su questa linea fa' si che il microprocessore legga dati dal registro del SID selezionato, mentre un basso sulla stessa linea consente al microprocessore di scrivere dati nel registro del SID selezionato. Questo Pin e' in genere connesso alla linea di lettura/scrittura del sistema.

CS (PIN 8)

Questo ingresso del livello TTL e' un selettore di circuito attivo basso che controlla il trasferimento dei dati tra il SID ed il microprocessore. CS deve essere basso per ogni trasferimento. Una lettura dal registro del SID selezionato puo' avvenire solo se CS e' basso, O2 e' alto e R/W e' alto, mentre la scrittura puo' avvenire solo se CS e' basso, O2 e' alto e R/W e' basso. Questo Pin e' di solito connesso all'insieme di circuiti di decodifica dell'indirizzo, in modo da permettere al SID di risiedere nella mappa di memoria del sistema.

A0–A4 (PIN 9–13)

Questi ingressi del livello TTL sono usati per scegliere uno dei 29 registri del SID. Anche se per la selezione di uno fra 32 registri sono forniti abbastanza indirizzi, le rimanenti 3 locazioni del registro non sono usate. Una scrittura su una di queste 3 locazioni e' ignorata; una lettura ritorna dati non validi. Questi Pin sono connessi di solito alle corrispondenti linee di indirizzo del microprocessore, in modo da consentire al SID lo stesso metodo di indirizzamento della memoria.

GND (PIN 14)

Per i migliori risultati, e' utile separate la linea di terra, tra il SID e l'alimentazione dalle linee di terra degli altri circuiti digitali. Questo collegamento minimizza il disturbo digitale sull'uscita audio.

D0–D7 (PIN 15–22)

Linee bidirezionali usate per il trasferimento di dati tra il SID e il microprocessore. Sono TTL-compatibili nel modo ingresso e capaci di pilotare due carichi TTL nel modo uscita. I buffer dei dati sono di solito nello stato alta impedenza esclusa. Durante un'operazione di scrittura, i buffer dei dati rimangono nello stato OFF (ingresso) ed il microprocessore fornisce dati al SID su queste linee. Durante un'operazionde di lettura, i buffer si portano su ON ed il SID fornisce dati al microprocessore su queste linee. I pin sono normalmente connessi alle corrispondenti linee di dati del microprocessore

POTX, POTY (PIN 24–23)

Ingressi ai convertitori A/D usati per rappresentare in forma digitale la posizione dei potenziometri. Il processo di conversione si basa sulla costante di tempo di un condensatore allacciato dal Pin POT a terra, quando venga caricato da un potenziometro allacciato dal Pin POT ad una tensione di +5 Volts. I valori dei componenti sono dati da:

RC = 4.7 E-4

essendo R la resistenza massima del POT e C il condensatore. Per R e C si consigliano valori rispettivamente di 470 kOhm e 1000 pF. Da notare che per ogni pin POT sono necessari uno zoccolo ed un POT separati.

Vcc (PIN 25)

Come con la linea GND, occorre stendere una linea separata a +5Vcc tra Vcc del SID e l'alimentazione, minimizzando cosi' il rumore di fondo. In prossimita' del pin e' necessario posizionare un condensatore di bypass.

EXT IN (PIN 26)

Ingresso analogico che consente la miscelazione di segnali audio esterni con l'uscita audio del SID, o la loro rielaborazione

attraverso il Filtro. Tipiche sorgenti sono la voce, la chitarra e l'organo. L'impedenza di ingresso di questo pin e' di circa 100 kOhm. Qualunque segnale applicato direttamente al pin deve viaggiare ad un livello di 6 Vcc, senza superare i 3 V p-p. Per evitare interferenze causate da differenze del livello di corrente continua, i segnali esterni devono essere accoppiati in c.a. a EXT IN per mezzo di un condensatore elttrolitico di capacita' 1...10 nF. Poiche' il percorso audio diretto (FILTEX=0) presenta un guadagno unitario, EXT IN puo' essere usato per miscelare le uscite di piu' circuiti SID, disponendo per tali circuiti un allacciamento "daisy-chain". Il numero di circuiti allacciabili in questo modo e' determinato dalla quantita' di rumore e distorsione consentito all'uscita finale. Da notare che il controllo uscita del Volume non influisce solamente sulle tre voci del SID, ma anche su qualunque ingresso esterno.

AUDIO OUT (PIN 27)

Questo buffer aperto alla sorgente e' l'uscita finale audio del SID, comprese le tre voci del Sid, il Filtro e qualunque ingresso esterno. Il livello di uscita viene impostato dal controllo uscita del Volume e raggiunge un massimo di 2 V p-p ad un livello di 6 Vcc. Per un corretto funzionamento e' necessario un resistore sorgente da AUDIO OUT a terra; per un'impedenza di uscita standard si consiglia una resistenza da 1 kOhm.

Poiche' l'uscita del SID viaggia ad un livello di 6 Vcc, deve essere accoppiata in c.a. a qualunque amplificatore audio con un condensatore elettrolitico di capacita' 1...10 nF.

Vdd (PIN 28)

Come per Vcc, occorre stendere una linea separata a +12 Volts cc ed usare un condensatore di bypass.

CARATTERISTICHE DEL 6581 SID

VALORI MASSIMI

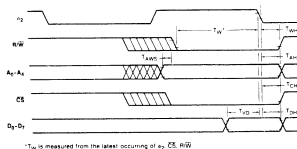
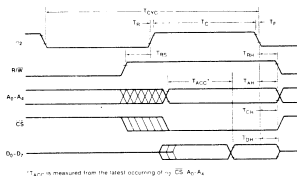
QUANTITÀ	SIMBOLO	VALORE	UNITÀ
Alimentazione	Vdd	-0.3...+17	VDC
Alimentazione	Vcc	-0.3....+7	VDC
Voltaggio ingresso (analogico)	Vina	-0.3...+17	VDC
Voltaggio ingresso (digitale)	Vind	-0.3....+7	VDC
Temperatura di funzionamento	Ta	0...+70	C
Temperatura di registrazione	Tstg	-55...+150	C

CARATTERISTICHE ELETTRICHE
(Vdd = 12 VDC ± 5%, Vcc = 5 VDC ± 5%, Ta = 0...70GC)

CARATTERISTICA	SIMBOLO	MIN	TIP	MAX	UNITÀ
Tensione alta di ingresso (RES,O2,R/W,CS,A0-A4,D0-D7)	Vih	2	-	Vcc	Vdc
Tensione bassa di ingresso (RES,O2,R/W,CS,A0-A4,D0-D7)	Vil	-0.3	-	0.8	Vdc
Perdita di corrente all'ingresso (RES,O2,R/W,CS,A0-A4) Vin=0...5 Vdc	Iin	-	-	2.5	nA
Tre Stati (OFF) (D0-D7) Vcc=max	Itsi	-	-	10	nA
Perdita di corrente all'ingresso Vin=0.4...2.4Vdc					
Tensione alta di uscita (D0-D7) Vcc=min Iload=200nA	Voh	2.4	-	Vcc-0.7	Vdc
Tensione bassa di uscita (D0-D7) Vcc=max Iload=3.2mA	Vol	GND	-	0.4	Vdc
Corrente alta di uscita (D0-D7);Sourcing Voh=2.4VDC	Ioh	200	-	-	nA
Corrente bassa di uscita (D0-D7);Sinking Vol=2.4VDC	Iol	3.2	-	-	mA
Capacita' di ingresso (RES,O2,R/W,CS,A0-A4,D0-D7)	Cin	-	-	10	pF
Voltaggio di POT trigger (POTX.POTY)	Vpot	-	Vcc/2	-	Vdc
Corrente di POT sink (POTX.POTY)	Ipot	500	-	-	nA
Impedenza di ingresso (EXT IN)	Rin	100	-	150	kOhm
Voltaggio di ingresso audio (EXT IN)	Vin	5.7	6.3	6	Vdc
Voltaggio di uscita audio (AUDIO OUT) 1 kOhm;load,volume=max	Vout	5.7	6.3	6	Vdc
a) Una Voce	Vout	0.4	0.6	0.5	Vac
b) Tutte le Voci	Vout	1.0	2.0	1.5	Vac
Corrente di alimentazione (Vdd)	Idd	-	20	25	mA
Corrente di alimentazione (Vcc)	Icc	-	70	100	mA
Dissipazione (totale)	Pd	-	1000	600	mW

TEMPORIZZATORE DEL SID 6581

*T_{ACC} is measured from the latest occurring of ϕ_2, $\overline{CS}$, A_0-A_4

*T_W is measured from the latest occurring of ϕ_2, $\overline{CS}$, $R/\overline{W}$

CICLO LETTURA

SIMBOLO	NOME	MIN	TIP	MAX	UNITÀ
Tcyc	Tempo ciclo del clock	1	-	20	usec
Tc	Ampiezza Pulsazione alta di clock	450	500	10000	nsec
Tr,Tf	Tempo salita/discesa del clock	-	-	25	nsec
Trs	Tempo impostazione	0	-	-	nsec
Trh	Tempo trattenimento lettura	0	-	-	nsec
Tacc	Tempo di accesso	-	-	300	nsec
Tah	Tempo trattenimento indirizzo	10	-	-	nsec
Tch	Tempo trattenimento scelta di circuito	0	-	-	nsec
Tdh	Tempo trattenimento dati	20	-	-	nsec

CICLO SCRITTURA

SIMBOLO	NOME	MIN	TIP	MAX	UNITÀ
Tw	Ampiezza pulsazione di scrittura	300	-	-	nsec
Twh	Tempo trattenimento scrittura	0	-	-	nsec
Taws	Tempo impostazione indirizzo	0	-	-	nsec
Tah	Tempo trattenimento indirizzo	10	-	-	nsec
Tch	Tempo trattenimento scelta di circuito	0	-	-	nsec
Tvd	Convalida dati	80	-	-	nsec
Tdh	Tempo trattenimento dati	10	-	-	nsec

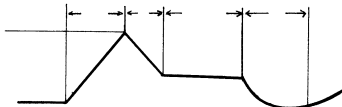
VALORI DELLA SCALA MUSICALE DI UGUALE ACCORDO

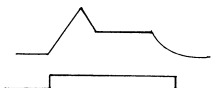
La tabella dell'Appendice E elenca i valori numerici che devono essere riportati nei registri di controllo della frequenza dell'Oscillatore del SID per la produzione di note aventi scala musicale di uguale accordo. La scala di uguale accordo e' formata da un'ottava contenente 12 semitoni (note): DO, RE, MI, FA, SOL ,LA,SI e DO#, RE#, FA#, SOL#, LA#. La frequenza di ogni semitono e' data dal prodotto della frequenza del semitono precedente per la radice dodicesima di due. La tabella si basa su un clock O2 oscillante a 1.02 MHz. Per l'uso di altre frequenze del clock master si rimanda all'equazione data nella Descrizione dei Registri (cfr. par. O.5). La scala scelta e' il tono fondamentale di concerto, in cui LA-4=440 Hz. Sono pure consentite trasposizioni di questa scala e di scale diverse da quella di uguale accordo.

Il metodo proposto nell'Appendice E per la generazione della scala di uguale accordo, sebbene semplice e veloce, risulta tuttavia inefficiente dal punto di vista della memoria occupata, in quanto per la sola tabella sono necessari 192 bytes. Si puo' migliorare l'efficienza della memoria determinando il valore della nota per mezzo di un algoritmo. Se si considera il fatto che ogni nota di un'ottava ha una frequenza pari esattamente alla meta' della frequenza della stessa nota nell'ottava successiva, allora l'intera tabella delle note puo' essere ridotta da 96 a 12 registrazioni, in quanto ci sono 12 note per ottava. Se queste 12 registrazioni (24 bytes) sono formate dai valori a 16 bit dell'ottava ottava (da DO7 a SI7), allora si possono ottenere le note delle ottave piu' basse scegliendo una nota nell'ottava ottava e dividendo il valore a 16 bit per 2 per ogni ottava di differenza. E poiche' la divisione per due altro non e' che uno scorrimento (shift) a destra del valore, questo calcolo puo' essere realizzato da una semplice routine del software. Anche se la nota SI7 oltrepassa l'intervallo degli oscillatori, il suo valore, per fini di calcolo, deve essere incluso lo stesso nella tabella (l'MSB di SI7 genera per il software un caso particolare, come la generazione di questo bit nel RIPORTO prima dello scorrimento). Ciascuna nota deve essere specificata in una forma indicante quale dei 12 semitoni si desidera, ed in quale delle otto ottave si trova tale semitono. Poiche' sono necessari 4 bit per la scelta di uno dei 12 semitoni e 3 bit per una delle 8 ottave, l'informazione complessiva rientra in un byte, in cui il semibyte basso seleziona il semitono (indirizzando l'intera tabella) ed il semibyte alto viene utilizzato dalla routine di divisione per determinare il numero di scorrimenti a destra del valore della tabella.

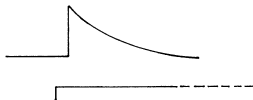
GENERATORI DI SVILUPPO DEL SID

Con il generatore di inviluppo a quattro parti ADSR (ATTACCARE, DECADERE, SOSTENERE, RILASCIARE) si e' provato, nella musica elettronica, a fornire il rapporto ottimo tra la flessibilita' e la semplicita' di controllo dell'ampiezza. Un'appropriata selezione dei

parametri di inviluppo permette la simulazione di una vasta gamma di strumenti a percussione e di accompagnamento. Un buon esempio di strumento di accompagnamento e' il violino. Il violinista controlla il volume attraverso la pressione dell'archetto sullo strumento. Tipicamente, il volume cresce lentamente, raggiunge un picco, poi ridiscende ad un livello intermedio. Il violinista puo' mantenere questo livello per tutto il tempo desiderato, poi il volume scende lentamente fino a scomparire. Un'"istantanea" di questo inviluppo puo' essere la seguente:

Questo inviluppo del volume puo' essere facilmente riprodotto dall'ADSR come illustrato sotto, con tipici valori di inviluppo:

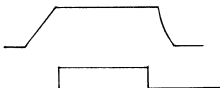
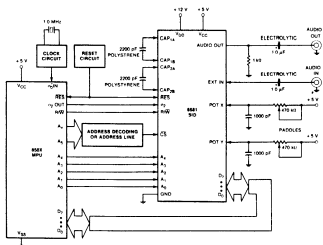
ATTACCARE:	10($A)	500 msec
DECADERE:	8	300 msec
SOSTENERE:	10($A)	
RILASCIARE:	9	750 msec

Da notare che il tono puo' essere tenuto al livello di SOSTENERE intermedio per quanto si desidera. Il tono inizia a scomparire non appena GATE viene azzerato. Con minori modifiche, questo inviluppo base puo' essere riutilizzato per ottoni e strumenti a fiato e a corda.

Un inviluppo completamente diverso viene prodotto da strumenti a percussione quali tamburi, piatti e gong, cosi' come da alcune tastiere come pianoforti e clavicembali. L'inviluppo delle percussioni e' caratterizzato da un ATTACCARE quasi istantaneo, seguito immediatamente da un DECADERE a volume zero. Gli strumenti a percussione non possono essere accompagnati ad un'ampiezza costante. Ad esempio, nel momento in cui viene battuto un tamburo, il suono raggiunge il massimo volume e decade rapidamente senza alcun riguardo al modo in cui il tamburo e' stato battuto. Un tipico inviluppo di piatto e' il seguente:

ATTACCARE:	0	2 msec
DECADERE:	9	750 msec
SOSTENERE:	0	
RILASCIARE:	9	750 msec

Da notare che il tono inizia a decadere a zero immediatamente dopo che e' stato raggiunto il valore di picco, senza curarsi di quando GATE e' stato azzerato. L'inviluppo di ampiezza di pianoforti e

clavicembali e' per qualche aspetto piu' complicato, ma puo' essere generato piuttosto facilmente per mezzo dell'ADSR. Questi strumenti raggiungono il massimo volume quando si preme per la prima volta un tasto. L'ampiezza inizia subito a decrescere lentamente, proseguendo per tutto il tempo che il tasto rimane premuto. Se il tasto viene rilasciato prima che il suono sia scomparso del tutto, l'ampiezza cade immediatamente a zero. Si ha quindi il seguente inviluppo:

Da notare che il tono decade lentamente fino all'azzeramento di GATE, dopodiche' l'ampiezza scende rapidamente a zero.
L'inviluppo piu' semplice e' quello dell'organo. Quando si preme un tasto, il tono raggiunge immediatamente il massimo volume, rimanendovi. Quando il tasto e' rilasciato, il tono cade immediatamente a zero. L'inviluppo e' il seguente:

La vera potenza del SID sta nella capacita' di creare suoni originali piuttosto che simulazioni di strumenti acustici. L'ADSR e' in grado di creare inviluppi che non corrispondono ad alcuno strumento "reale"; tipico esempio e' l'inviluppo "backwards", caratterizzato da un ATTACCARE lento ed un DECADERE rapido. Il suono che si ottiene e' per qualche verso simile ad uno strumento registrato su nastro e quindi suonato all'indietro. L'inviluppo che si genera e' il seguente:

Si possono creare molti suoni unici applicando l'inviluppo di ampiezza di uno strumento alla struttura armonica di un altro. Cio' produce suoni simili a strumenti acustici familiari, tuttavia notevolmente diversi. In generale, il suono e' piuttosto soggettivo, e per raggiungere il suono desiderato sono necessarie alcune prove con diversi valori di inviluppo.

TIPICA APPLICAZIONE DEL SID 6581

APPENDICE P

GLOSSARIO

ADSR	Inviluppo di Attacare/Decadere/Sostenere/Rilasciare
Attaccare	Valore a cui una nota musicale raggiunge il volume di picco
Binario	Sistema di numerazione in base 2
Booleani,Operatori	Operatori logici
Byte	Locazione di memoria
CHROMA,disturbo	Distorsione del colore
CIA	Complex Interface Adapter (Adattatore Interfaccia Complessa)
Coda	Linea a fila singola
DDR	Data Direction Register (Registro Direzione Dati)
Decadere	Valore a cui una nota musicale scende dal volume di picco al volume di Sostenere
Decimale	Sistema di numerazione in base 10
E	Costante matematica (circa 2.71828183)
Esadecimale	Sistema di numerazione in base 16
FIFO	First In / First Out
Intero	Numero senza la virgola decimale
Inviluppo	Intensita' del volume di una nota ad un certo istante
Jiffy Clock	Timer hardware di intervallo(Jiffy=1/60 di secondo)
NMI	Non-Maskable Interrupt (Interruzione Non Mascherabile)
Ottale	Sistema di numerazione in base 8
Operando	Parametro
OS	Operating System (Sistema Operativo)
Pixel	Punto di risoluzione dello schermo
Registro	Particolare locazione di registrazione in memoria
Rilasciare	Valore a cui una nota musicale scende dal volume di Sostenere a zero
ROM	Read Only Memory (Memoria a Sola Lettura)
SID	Sound Interface Device (Dispositivo Interfaccia del Suono)
Segnati,Numeri	Numeri positivi e negativi
Sintassi	Struttura delle frasi di programmazione
Sostegno	Livello di volume per sostenere una nota musicale
Sottoscritto	Variabile indice
Troncato	Tagliato, eliminato (non arrotondato)
VIC II	Video Interface Chip (Circuito Interfaccia Video)
Video,schermo	Televisore

PRONTUARIO DEL COMMODORE 64

VARIABILI SEMPLICI

Tipo	Nome	Intervallo
Reale	XY	+-1.70141183E+38 +-2.93873588E-39
Intera	XY%	+-32767
Stringa	XY$	0...255 caratteri

X e' una lettera (A-Z), Y e' una lettera o un numero (0-9). I nomi delle variabili possono essere lunghi piu' di due caratteri, ma solo i primi due sono riconosciuti.

VARIABILI SCHIERA

Tipo	Nome
Singola Dimensione	XY(5)
Due Dimensioni	XY(5,5)
Tre Dimensioni	XY(5,5,5)

Si possono usare, senza DIMensionarle, schiere fino a 11 elementi (indici 0-10).

OPERATORI ALGEBRICI

= Assegna il valore ad una variabile
- Negazione
   Esponenziazione
* Moltiplicazione
/ Divisione
+ Addizione
- Sottrazione

OPERATORI LOGICI E RELAZIONALI

=   Uguale
<> Non Uguale
<   Minore
>   Maggiore
<= Minore o Uguale
=> Maggiore o Uguale
NOT Negazione Logica
AND Congiunzione Logica
OR Disgiunzione Logica

1=Espressione VERA, 0=Espressione FALSA

COMANDI DI SISTEMA

LOAD "<nome>"	Carica un programma da cassetta
SAVE "<nome>"	Salva un programma su cassetta

LOAD "<nome>",8	Carica un programma da disco
SAVE "<nome>",8	Salva un programma su disco
VERIFY "<nome>"	Verifica che il salvataggio di un programma sia avvenuto senza errori
RUN	Esegue un programma
RUN xxx	Esegue un programma a partire dalla linea indicata
STOP	Interrompe l'esecuzione
END	Termina l'esecuzione
CONT	Continua l'esecuzione di un programma dalla linea a cui era stato fermato
PEEK(X)	Ritorna il contenuto della locazione di memoria X
POKE X,Y	Cambia il contenuto della locazione X con il valore Y
SYS xxxx	Salta all'esecuzione di un programma in linguaggio macchina a partire da xxxx
WAIT X,Y,Z	Il programma attende che il contenuto della locazione X, disgiunto (OR) con Y e congiunto (AND) con Z, sia diverso da zero
USR(X)	Passa ad una subroutine in linguaggio macchina il valore di X

COMANDI DI EDITING E DI FORMATTAZIONE

LIST	Lista l'intero programma
LIST A-B	Lista il programma dalla linea A alla B
REM <messaggio>	Si puo' scrivere un messaggio di commento, che pero' viene ignorato durante l'esecuzione del programma
TAB(X)	Usato in istruzioni PRINT; spazia di X posizioni sullo schermo
SPC(X)	Stampa X spazi su una linea
POS(X)	Ritorna la posizione corrente del cursore
CLR/HOME	Posiziona il cursore nell'angolo sinistro dello schermo
SHIFT CLR/HOME	Azzera lo schermo e posiziona il cursore nella posizione "HOME"
SHIFT INST/DEL	Inserisce uno spazio nella corrente posizione del cursore
CTRL	Usato con un tasto numerico di colore, sceglie il colore di testo. Puo' essere usato in istruzioni PRINT
CRSR <tasto>	Muove il cursore sullo schermo a destra, sinistra, alto, basso
Tasto Commodore	Usato con SHIFT, sceglie il modo lettere maiuscole/minuscole oppure il modo grafico

SCHIERE E STRINGHE

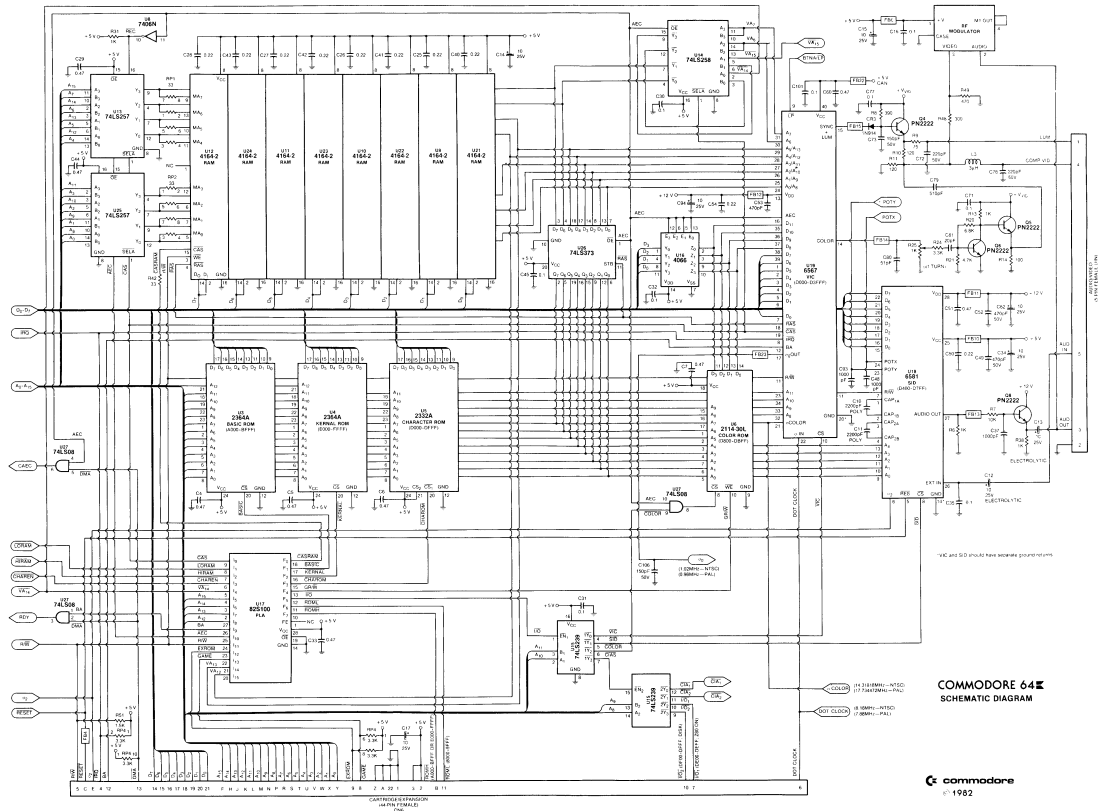
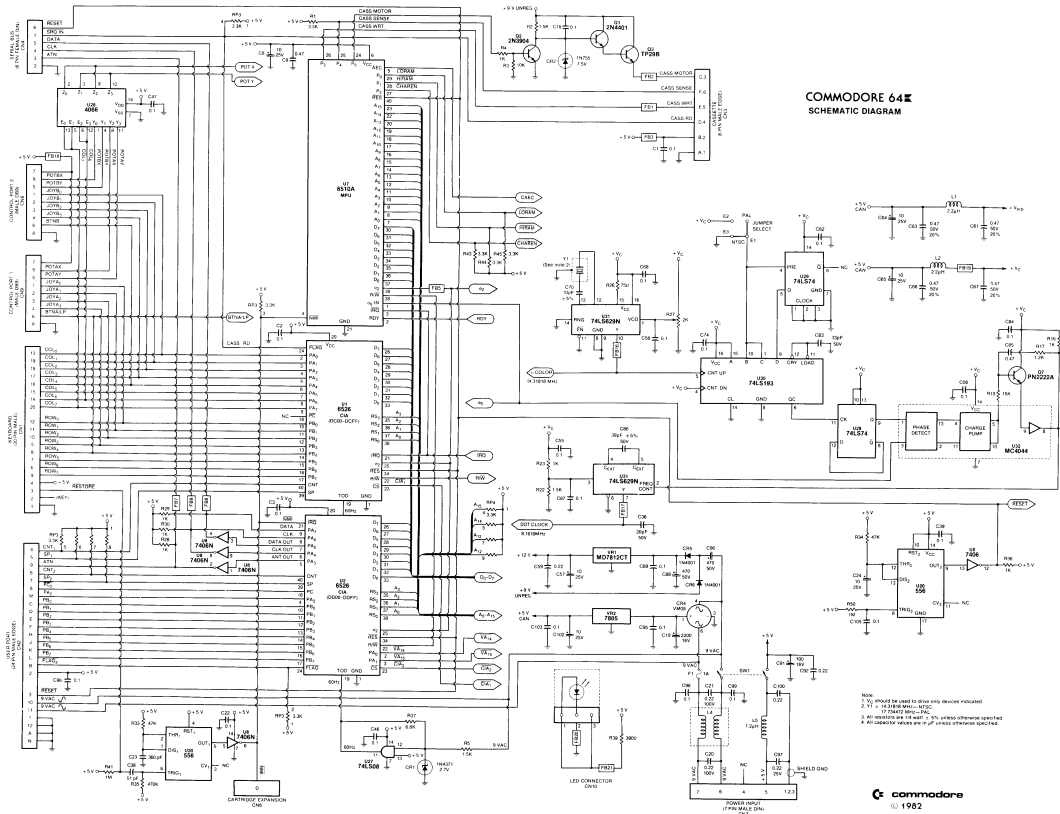
DIM A(X,Y,Z)	Imposta al massimo gli indici di A; riserva spazio per un totale di (x+1)*(Y+1)*(Z+1) elementi, iniziando da A(0,0,0)
LEN(X$)	Ritorna il numero di caratteri di X$
STR$(X)	Ritorna il valore numerico di X convertito in una stringa
VAL(X$)	Ritorna il valore numerico di X$, fino al primo carattere non numerico

CHR$(X)	Ritorna il carattere ASCII il cui codice e' X
ASC(X$)	Ritorna il codice ASCII del primo carattere di X$
LEFT$(A$,X)	Ritorna gli X caratteri piu' a sinistra di A$
RIGHT$(A$,X)	Ritorna gli X caratteri piu' a destra di A$
MID$(A$,X,Y)	Ritorna Y caratteri di A$ a partire dal carattere X

COMANDI DI INPUT/OUTPUT

INPUT A$ (o A)	Scrive "?" sullo schermo ed attende che l'Utente invii una stringa o un valore
INPUT "ABC";A	Scrive il messaggio ed attende che l'Utente invii un valore. E' ammesso anche INPUT A$
GET A$ (o A)	Attende che l'Utente prema un tasto; non occorre RETURN
DATA A,"B",C	Inizializza un insieme di valori che possono essere usati da un'istruzione READ
READ A$ (o A)	Assegna ad A$ (o A) il valore della prossima DATA
RESTORE	Riposiziona il puntatore dati all'inizio della lista DATA per una nuova lettura
PRINT "A=";A	Scrive la stringa "A=" seguita dal valore di A; ";" sopprime gli spazi, "," tabula i dati a partire dal prossimo campo

FLUSSO DEL PROGRAMMA

GOTO X	Salta alla linea X
IF A=3 THEN 10	Se A=3 e' VERO esegue la rimanente parte della istruzione, altrimenti esegue la linea successiva
FOR A=1 TO 10 STEP2:NEXT	Esegue tutte le istruzioni comprese tra FOR ed il corrispondente NEXT; A va da 1 a 10 con passo 2
NEXT A	Indica la fine di un ciclo. A e' opzionale
GOSUB 2000	Salta alla subroutine che inizia alla linea 2000
RETURN	Indica la fine della subroutine. Ritorna alla linea dell'istruzione successiva all'ultimo GOSUB
ON X GOTO A,B	Salta all'X-esimo numero di linea della lista. Se X e' 1 salta ad A, ecc.
ON X GOSUB A,B	Salta alla subroutine che inizia all'X-esimo numero di linea della lista

COMMODORE 64
SCHEMATIC DIAGRAM
U7 6510A MPU
U1 6526 CIA (DC00-DCFF)
U2 6526 CIA (DD00-DDFF)
U28 4066
U31 74LS629N
U29 74LS74
U30 74LS193
U32 MC4044
U20 556
U27 74LS08
VR1 MD7812CT
VR2 7805
SERIAL BUS (6 PIN FEMALE DIN) CN4
CONTROL PORT 2 (MALE DB9) CN8
CONTROL PORT 1 (MALE DB9) CN9
KEYBOARD (20 PIN MALE) CN1
USER PORT (24-PIN MALE EDGE) CN2
CASSETTE (6 PIN MALE EDGE) CN3
CARTRIDGE EXPANSION CN6
LED CONNECTOR CN10
POWER INPUT (7-PIN MALE DIN) CN7
Note:
1. V_C should be used to drive only devices indicated.
2. Y1 = 14.31818 MHz—NTSC.
17.734472 MHz—PAL.
3. All resistors are 1/4 watt ± 5% unless otherwise specified.
4. All capacitor values are in μF unless otherwise specified.
commodore
© 1982